Miscellaneous Paper W-97-1
August 1997

US Army Corps of Engineers
Waterways Experiment Station

Water Quality Research Program

Development of Unstructured Grid Linkage Methodology and Software for CE-QUAL-ICM

by Raymond S. Chapman, Ray Chapman and Associates
Terry K. Gerald, AScI, Inc.
Mark S. Dortch, WES

Prepared for Headquarters, U.S. Army Corps of Engineers

Water Quality
Research Program

Miscellaneous Paper W-97-1
August 1997

Development of Unstructured Grid Linkage Methodology and Software for CE-QUAL-ICM

by Raymond S. Chapman

Ray Chapman and Associates
1725 MacArthur Place
Vicksburg, MS 39180

Terry K. Gerald

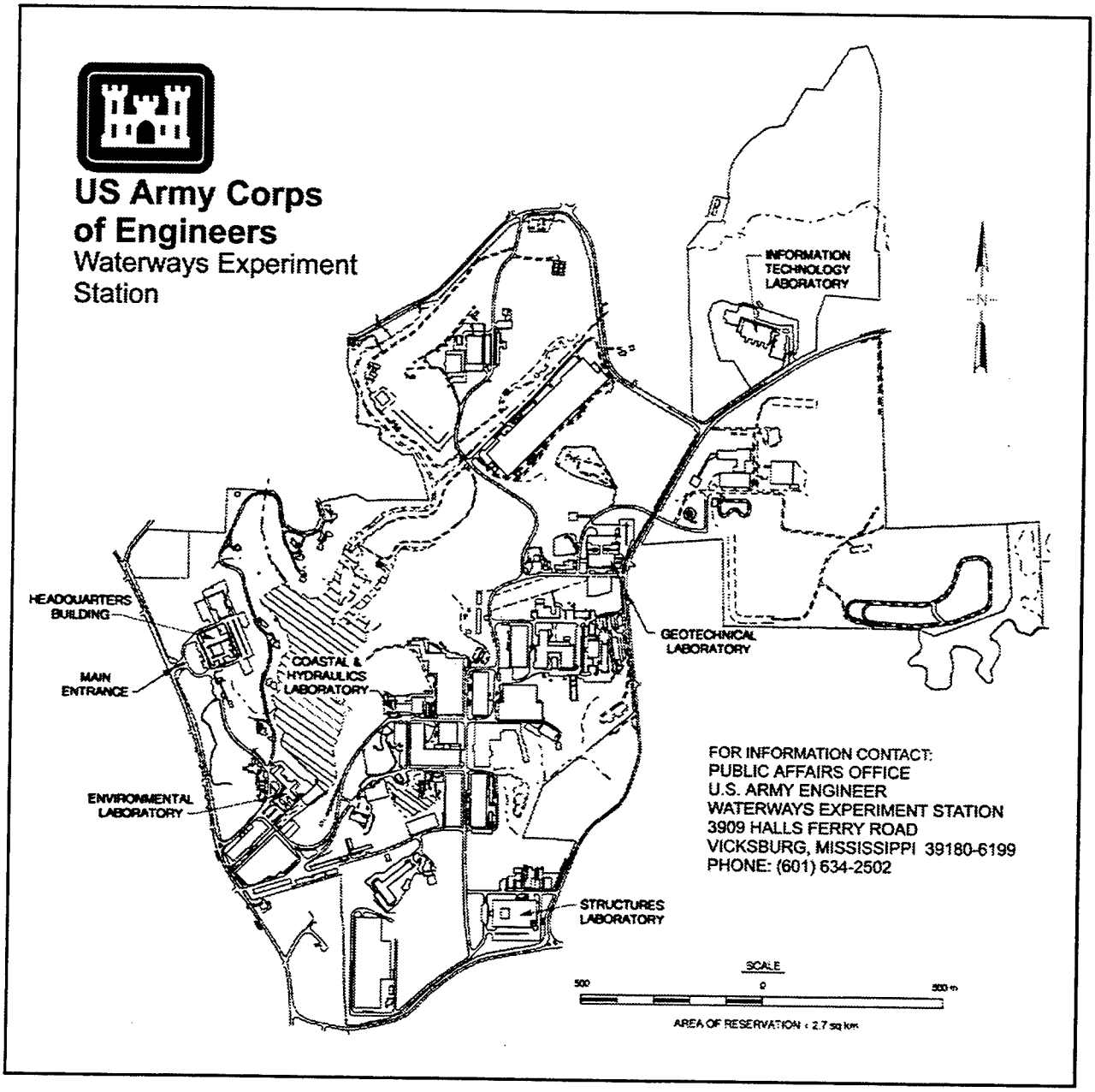
AScI Corporation
3402 Wisconsin Avenue
Vicksburg, MS 39180

Mark S. Dortch

U.S. Army Corps of Engineers
Waterways Experiment Station
3909 Halls Ferry Road
Vicksburg, MS 39180-6199

Final report

Prepared for U.S. Army Corps of Engineers
Washington, DC 20314-1000



Waterways Experiment Station Cataloging-in-Publication Data

Chapman, Raymond S.

Development of unstructured grid linkage methodology and software for CE-QUAL-ICM / by Raymond S. Chapman, Terry K. Gerald, Mark S. Dortch ; prepared for U.S. Army Corps of Engineers.

185 p. : ill. ; 28 cm. — (Miscellaneous paper ; W-97-1)

Includes bibliographic references.

1. Hydrologic models — Software. 2. Water quality — Software. 3. Environmental quality —Software. 4. Water — Composition — Software. I. Gerald, Terry K. II. Dortch, Mark S. III. United States. Army. Corps of Engineers. IV. U.S. Amy Engineer Waterways Experiment Station. V. Water Quality Research Program (U.S.) VI. Title. VII. Miscellaneous paper (U.S. Army Engineer Waterways Experiment Station) ; W-97-1.

TA7 W34m no.W-97-1

Contents

List of Figures

Preface

The work reported herein was conducted as part of the Water Quality Research Program (WQRP), Work Unit 32808, "Model of Contaminant Transport and Fate at Corps Projects." The WQRP is sponsored by the Headquarters, U.S. Army Corps of Engineers (HQUSACE), and is assigned to the U.S. Army Engineer Waterways Experiment Station (WES) under the purview of the Environmental Laboratory (EL). Funding was provided under Department of the Army Appropriation 96X3121, General Investigation. The WQRP is managed under the Environmental Modeling, Simulation, and Assessment Center (EMSAC), Dr. John W. Barko, Director for EL. Mr. Robert C. Gunkel was Assistant Manager for the WQRP. Program Monitor during this study was Mr. Frederick B. Juhle, HQUSACE.

The Principal Investigator of Work Unit 32808 was Dr. Mark S. Dortch, Chief, Water Quality and Contaminant Modeling Branch (WQCMB), Environmental Processes and Effects Division (EPED), EL. The work reported herein was conducted by Dr. Raymond S. Chapman, Ray Chapman and Associates, Vicksburg, MS, and Mr. Terry K. Gerald, AScI, Inc., Vicksburg, MS, under contract to WES. This report was prepared by Dr. Chapman and Mr. Gerald. Dr. Dortch monitored the contract and reviewed, edited, and revised the report.

This study was conducted under the general supervision of Dr. Richard E. Price, Acting Chief, EPED, and Dr. John Harrison, Director, EL. This report was reviewed by Messrs. Thomas Cole and Ross Hall, WQCMB.

At the time of publication of this report, Director of WES was Dr. Robert W. Whalin.

This report should be cited as follows:

Chapman, R. S., Gerald, T. K., and Dortch, M. S. (1997). "Development of unstructured grid linkage methodology and software for CE-QUAL-ICM," Miscellaneous Paper W-97-1, U.S. Army Engineer Waterways Experiment Station, Vicksburg, MS.

1 Introduction

Background

CE-QUAL-ICM (Cerco and Cole 1995) is a three-dimensional (3-D) water quality model based on the finite volume approach. There are two versions of this model, a eutrophication version documented by Cerco and Cole (1995), and a toxic substance version referred to as ICM/TOXI documented by Wang et al. (1996). The project that funded the development of ICM/TOXI also funded the present study reported herein.

Neither version of CE-QUAL-ICM (ICM) solves for hydrodynamics, so this information must be supplied by a hydrodynamic model for driving transport in the ICM model. The ICM model is usually linked to output from CH3D-WES (Johnson et al. 1991), a 3-D finite difference hydrodynamic model based on a structured grid scheme. Structured grids assume that computational cells are ordered along columns and rows, whereas unstructured grids allow cells to be arranged in a nonordered fashion. Thus, unstructured grids allow greater flexibility for describing the geometry of the domain.

RMA10 (Norton, King, and Orlob 1973; Thomas and McAnally 1990) is a 3-D finite element hydrodynamic model frequently used by the U.S. Army Engineer Waterways Experiment Station (WES). RMA10 is based on an unstructured grid approach. There was a need for developing a linkage to the RMA10 model to allow greater flexibility provided by unstructured grids for conducting water quality and contaminant model studies.

Objective and Scope

The objective of this study was to develop software to provide linkage of RMA10 output to the ICM code and to test the success of the linkage for simple test cases. The software development consisted of basically three parts: (a) development of a code (MAPPER) to map the finite element grid configuration and geometry information into a file that can be interpreted by the ICM code; (b) development of an RMA10 postprocessor code (FEMCONVT) to convert RMA10 velocity and water surface information at nodes to flows and

cell areas and volumes for each ICM cell; and (c) modifications within ICM to conform to the RMA10 linkage. Linkage testing consisted of checking local and global volume and mass conservation and examining the transport against known solutions. It is important to maintain mass conservation in water quality models since they are based on the mass conservation principle.

This report is organized into chapters on Introduction, Implementation, Testing, and Summary and Recommendations. Appendixes A-C deal with the MAPPER, FEMCONVT, and ICM codes.

2 Implementation

The basic requirements for the design and development of an unstructured grid linkage methodology for ICM are as follows: (a) generating a geometry file that relates the elements and their geometric attributes for an unstructured grid hydrodynamic model, such as RMA10, to flow faces and computational cells of ICM; (b) assigning a tagging system to relate flows from the hydrodynamic model to ICM flow faces; (c) establishing a convention for positive/negative flow directions; and (d) computing integrated flows across and normal to each hydrodynamic element face. The linkage program MAPPER conducts the first three tasks, while the linkage program FEMCONVT conducts the last.

MAPPER

The linkage geometry and map files are derived from the RMA10 element connection file that is generated for a hydrodynamic run. A C language program (MAPPER.C) was written to generate this information. MAPPER.C is described and listed in Appendix A. Descriptions of the RMA10 node and element connectivity scheme were obtained from the RMA10 documentation (Thomas and McAnally 1990; Brigham Young University 1994). The element connection file, generated by an RMA10 preprocessor code, which resides within the WES Coastal and Hydraulics Laboratory (CHL), provides element and node numbering, the 3-D coordinates of each node within the grid, and boundary designation information. This file is read by MAPPER.C, which generates the geometry (GEO) and map (MAP) files required as input to ICM.

To accommodate unstructured grid hydrodynamics, the GEO file was modified so that a one-dimensional array of centroid-based grid distances, which are direction sensitive, is computed and replaces the original box lengths (i.e., BL(IB(F),QD(F))). The centroid-based grid lengths data consist of (a) distance from centroid of IB cell to flow face (BID); (b) centroid to centroid distance between IB and JB cells (BI); (c) centroid to centroid distance between the ILB cell and IB (BIL); and (d) centroid to centroid distance between the JB cell and JRB cell (BIR) (see Figure 1). The specification of the ILB, IB, JB, and JRB ICM cell numbers (Figure 1) is accomplished through examination of the element connection file. Specifically, when elements that share common mid-side nodes are identified, a cell face between boxes IB and JB is defined. Cells

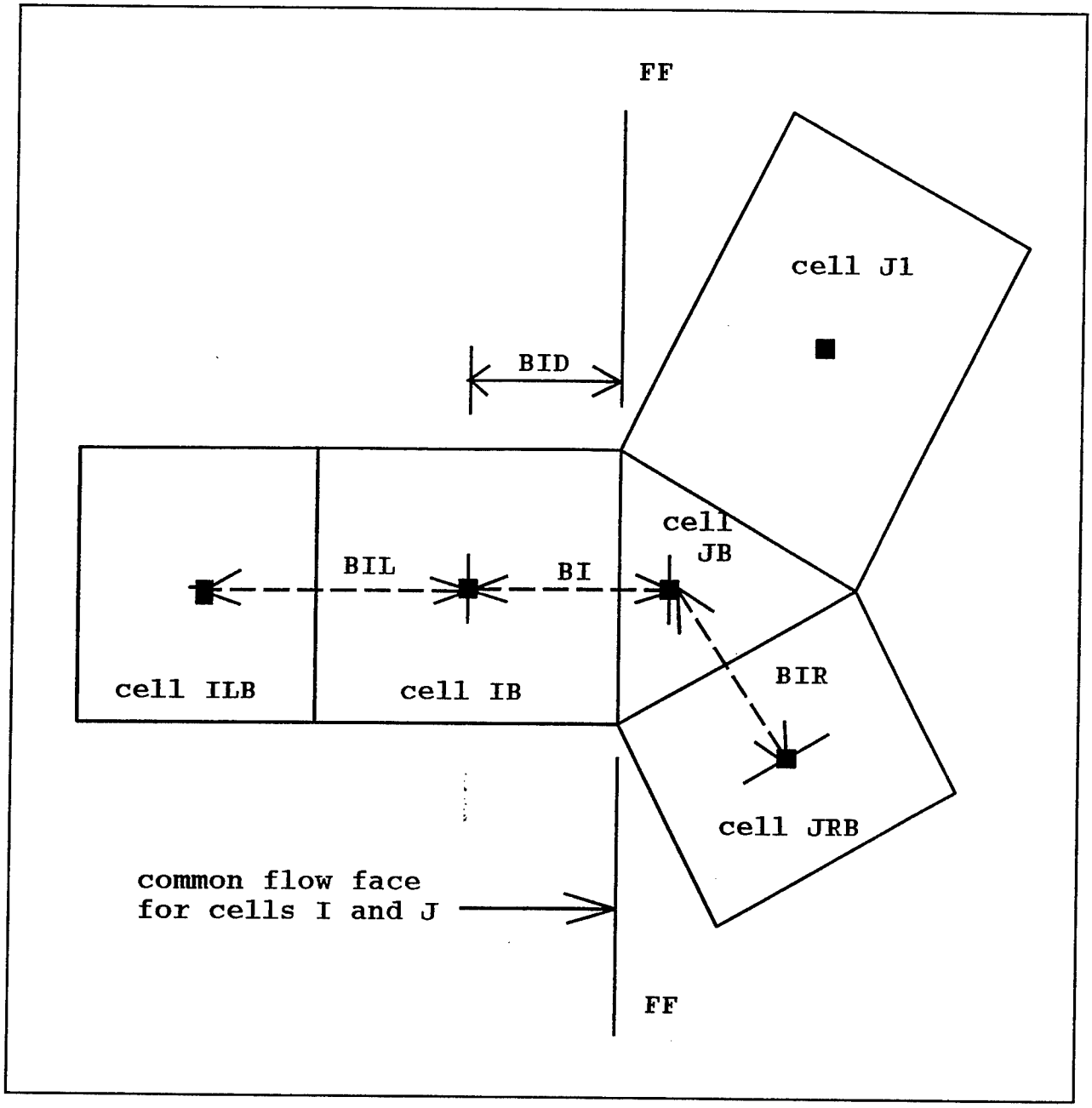


Figure 1. Grid nomenclature

to the right or left of cells with a common flow face are determined by selecting the cell with the shortest centroid distance. For example, as shown in Figure 1, the centroid distance from Cells JB to JRB is shorter than the distance from Cells JB to J1; thus, Cell JRB is selected as the adjacent right cell, or the far downstream cell for the QUICKEST weighting, and BIR is the distance from JB to JRB.

Due to the fact that unstructured grids have no preferred grid direction, a convention has been adopted. A flow direction for the establishment of the GEO

and MAP files is assumed based upon the cell numbering scheme obtained from RMA10. In the horizontal direction, flow is defined to be positive when it passes from a smaller RMA10 element number to a larger RMA10 element number. In the vertical direction, flows are defined positive upward, which is the convention used in both RMA10 and ICM. When the flow field is read in from the HYD file, the actual flow directions relative to the assumed flow directions are determined so that the proper cells are sampled for the QUICKEST advection operators.

The general concept of the MAP file was unaltered; however, additional information was required to relate the RMA10 and ICM grids. The processor program MAPPER assigns ICM cell numbers to RMA10 element numbers and writes the ordered pairs as the first set of records in the MAP file. The correspondence between hydrodynamic elements and ICM cells is arbitrary in that the connection is accomplished via flow face definitions. This approach is adopted so that the existing convention for numbering cells and flow faces in ICM is unaltered.

The RMA10 element edges and ICM flow faces are related in the MAP file via a tag. In the horizontal direction, the flow faces between adjacent cells in ICM are tagged to mid-side nodes in RMA10, which are defined along element edges. Vertical flow faces in ICM are tagged to the top of each element in RMA10. These tags are used via translation arrays to relate RMA10 flow locations to the corresponding sequentially numbered ICM flow face numbers.

Boundaries within the RMA10 grid are specified via nodal codes that define land, inflow, and tidal or head boundaries. These codes are used in MAPPER to specify boundaries in the MAP file.

FEMCONVT

The Fortran language program FEMCONVT.f was written by Dr. R. C. Berger of the WES CHL to read a binary RMA10 hydrodynamic output file and convert nodal velocities and surface elevations into cell face areas (including planar surface areas), volumes, and flows, which are written to another output file (HYD file). A listing of FEMCONVT is provided in Appendix B. The output of FEMCONVT is segregated into horizontal and vertical components. In the horizontal direction, the information provided is time-varying horizontal flow face areas and flows for each mid-side node. The vertical part of the hydrodynamic output provides element number, element planar surface area, volume, and the vertical flow defined on the top face of each element.

ICM Modifications

The original advection operator in ICM was designed to work with structured grid hydrodynamic models such as CH3D-WES (Chapman 1988). As a result, the grid is described by rows and columns of grid cells and box lengths. These

box lengths are used to compute the QUICKEST multipliers used for transport. Due to the success realized during numerous applications of the QUICKEST transport algorithm in both the ICM and CE-QUAL-W2 models (Dortch, Chapman, and Abt 1991; Chapman 1992; Chapman and Cole 1992), a similar approach was adopted for the unstructured version of ICM. The unstructured grid modifications made to the original version of ICM are presented in Appendix C. For ICM to accommodate unstructured hydrodynamic models such as RMA10, the QUICKEST multipliers had to be recast in terms of element centroid to centroid distances. As previously discussed, the linkage processor program MAPPER generates the four length scales needed to develop the QUICKEST multipliers. Using these length scales, the QUICKEST multipliers were rewritten and checked to ensure that the advection multipliers summed to one, and the diffusive multipliers summed to zero.

Linkage Procedure

To complete a linkage of RMA10 and ICM, the steps outlined below must be completed.

a. A binary output file generated by RMA10, which contains the time-invariant grid and time-varying hydrodynamic data, must be obtained from the hydrodynamic modeling team. The name of this file is specified as RMA10 input, so it is arbitrary.

b. With regard to the time-invariant grid data, a program named R4ICR10.f, written and provided by CHL, is used to read the RMA10 binary file and generate a 3-D ASCII element connection file named R4ICR10.out. This file describes the juxtaposition of all elements and nodes in the RMA10 grid.

c. The linkage program MAPPER.C reads R4ICR10.out and generates the ICM MAP and GEO files. The number and position of ICM cells or boxes are determined by MAPPER automatically. Presently, a parameter statement within the MAPPER code defines "number_of_layers," which must be set to the desired number of layers. This parameter can be used to limit the output to a single layer for debug purposes. When this parameter is set to the correct number of RMA10 layers, the complete MAP and GEO files are automatically generated. MAPPER input (i.e., R4ICR10.out) and output files for a simple 3 by 2 by 2 grid are presented in Appendix D.

d. The time-varying hydrodynamic data are also extracted from the same RMA10 binary output file. This is accomplished using the FEMCONVT.f program. This program has one interactive input, which is the RMA10 binary file name. All grid parameters are read from the RMA10 binary file. The output of this program is an ASCII ICM HYD file. At this time, FEMCONVT.f has several limitations:

- ASCII output should be written in binary to reduce space requirements for applications with large output files.
- Base elevation in feet relative to an RMA10 datum must be hardwired.
- FEMCONVT only works for quadrilateral elements.
- Parameter statements must be checked to ensure the code will accommodate the size of the grid.
- Vertical diffusivities required by ICM are not computed nor output.
- There is no provision for temporal or spatial averaging of hydrodynamic output.

e. Two additional parameters must be defined in the ICM file WQM_COM.INC. NFEMHFFP and NFEMVFFP are the maximum number of horizontal and vertical flow faces, respectively.

3 Testing

Volume Balance Testing

Initial testing of the linkage methodology using hydrodynamic flow information was performed on a simple 26 by 4 by 3 rectangular contracted channel comprised of quadrilaterals as shown in Figure 2. To test both local and global volume conservation, a single steady-state flow field was used. Given that the flow field has in fact converged to a steady-state solution, successive applications of the flow field using the steady-state distribution of element volumes as an initial state should result in no temporal change in volume. Using these data, ICM was run using a 1-hr hydrodynamic update for 0.2 days, or two hydrodynamic updates. As expected, global volume conservation was achieved. However, significant variations in local volume conservation were observed throughout the grid. This problem was further investigated by performing flow balances on individual cells. In regions where elements were orthogonal, flow imbalances on the order of 1 percent were found, where the error in flow balance is based on the difference in the sum of the flows scaled by the maximum flow in or out of that cell. In regions where elements were nonorthogonal, flow imbalances were on the order of 60 percent. Discussions with CHL suggested that this problem can be overcome or at least reduced to errors on the order of a few percent. Means of correcting the flow/volume imbalances are being pursued.

Given that RMA10 flow fields can be produced with small but possibly acceptable volume errors, two alternatives can be adopted that will result in a local volume balance. The simplest option is to use the flows from the RMA10 hydrodynamic processor, but not use the resulting volumes. Under this scenario, time-varying volumes are computed within ICM using the previously computed volume and the input flows, thereby ensuring volume and mass conservation. The downside of this option is that if a persistent net loss or gain is realized in one or more cells, problems could develop during long-term simulations. Specifically, cells could either dry out or become unrealistically large.

Alternatively, both RMA10 flows and volumes can be used by ICM. The RMA10 flow data would again be used within ICM to calculate conserving volumes for each hydrodynamic update interval. The difference in the RMA10 volumes and ICM volumes can then be used to determine the volume errors for

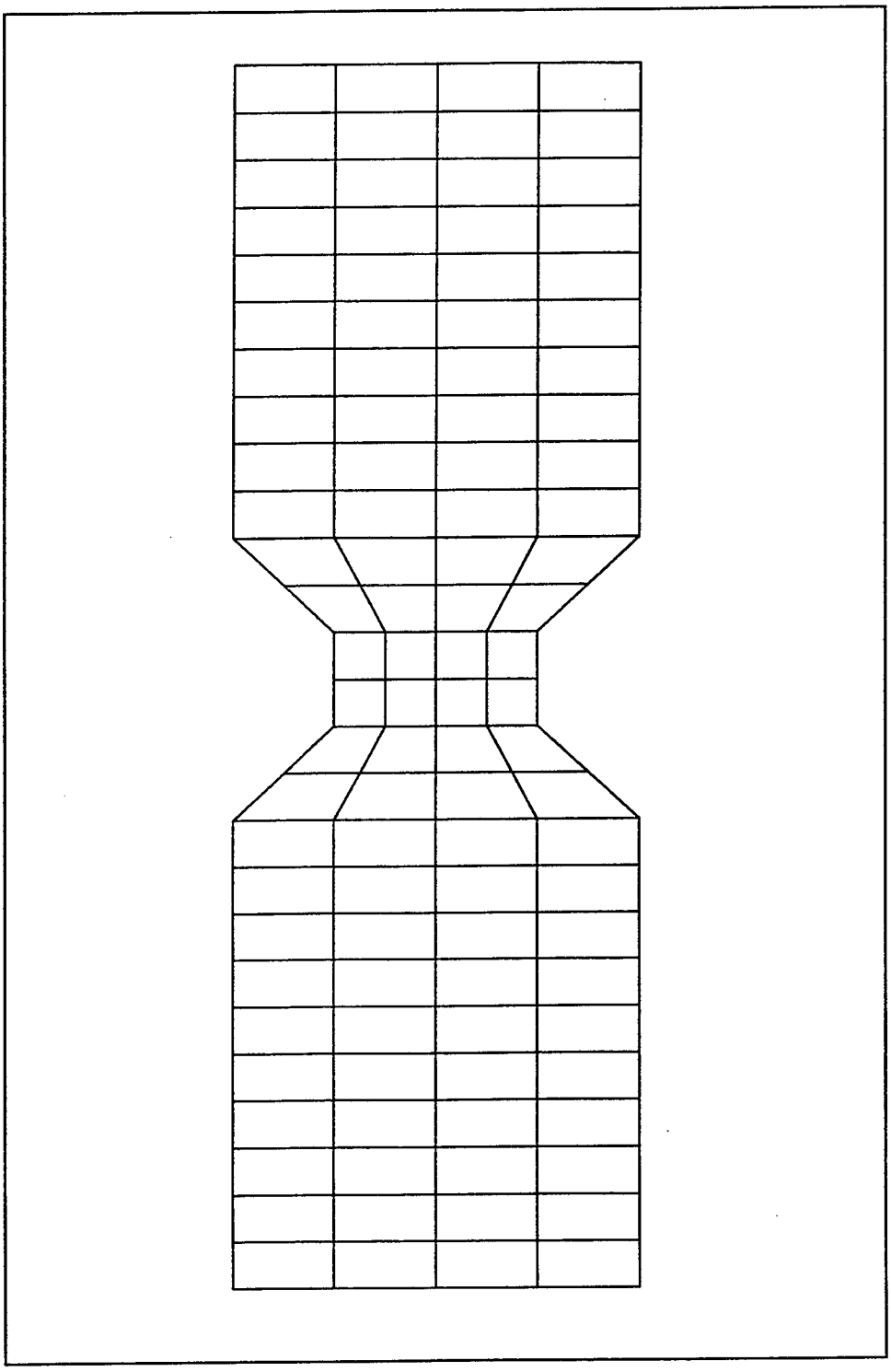


Figure 2. Volume balance test grid

each cell. If the volume error is ignored, i.e., RMA10 volumes are used, the model will not conserve mass, but concentrations will be conserved. For example, if a water quality constituent of 10 ppm is introduced into a bay that

contains 10 ppm, the concentrations will all remain 10 ppm, but some mass of the constituent will be gained or lost.

Volume and mass conservation can be forced when using both RMA10 flows and volumes by introducing a volume error correction. However, under this option the concentrations will not remain conservative, or 10 ppm everywhere as in the previous example. The error correction for each hydrodynamic update interval is computed and added (or subtracted) incrementally over the ICM time-steps within that update interval. Thus, the ICM volume computed at the end of each hydrodynamic update interval will be identical to the corresponding RMA10 volume. The simplest way to implement this correction is to generate a source flow by dividing the total volume error for each hydrodynamic update interval by the hydrodynamic update time interval. As the ICM simulation proceeds, a volume source/sink can be added/subtracted by multiplying the source flow by the ICM time-step.

Transport Testing

To test the revisions to ICM for the unstructured grid QUICKEST multipliers, transport testing was done using a one-dimensional rectangular grid set up without and with the unstructured grid multipliers as derived from the GEO and MAP files generated by MAPPER. The RMA10 model was not used for these tests. This grid consisted of 25 boxes 100 m in length with 1-m^2 cell face areas. A constant flow rate of 0.1 m^3/sec was specified with no horizontal diffusion. Specifying a constant source concentration at the inflow, the predicted square wave concentrations in both simulations (i.e., with structured and with unstructured grid) were identical as shown in Figures 3 and 4. The algorithm was tested for variable grid spacing by alternating between 100- and 50-m box lengths and repeating the simulations. Additionally, both positive and negative flows were tested. All runs yielded identical results, which confirmed correct implementation of QUICKEST.

Additional testing of transport was performed using a steady-state RMA10 hydrodynamic flow field. A rectangular channel RMA10 grid was used consisting of 10 elements along the channel, 4 elements across the channel, and 3 elements deep (see Figure 5). Output from RMA10 was used to drive ICM. Exact global mass conservation was verified via a simulation of a conservative tracer spot dump and simulation of a uniform tracer concentration throughout the grid and boundaries. In addition, a constant source square wave test was performed. Despite the coarseness of the grid, the expected characteristics of QUICKEST are preserved in that minimal undershoot and overshoot and smearing of the wave occur over about five grid points (see Figure 6).

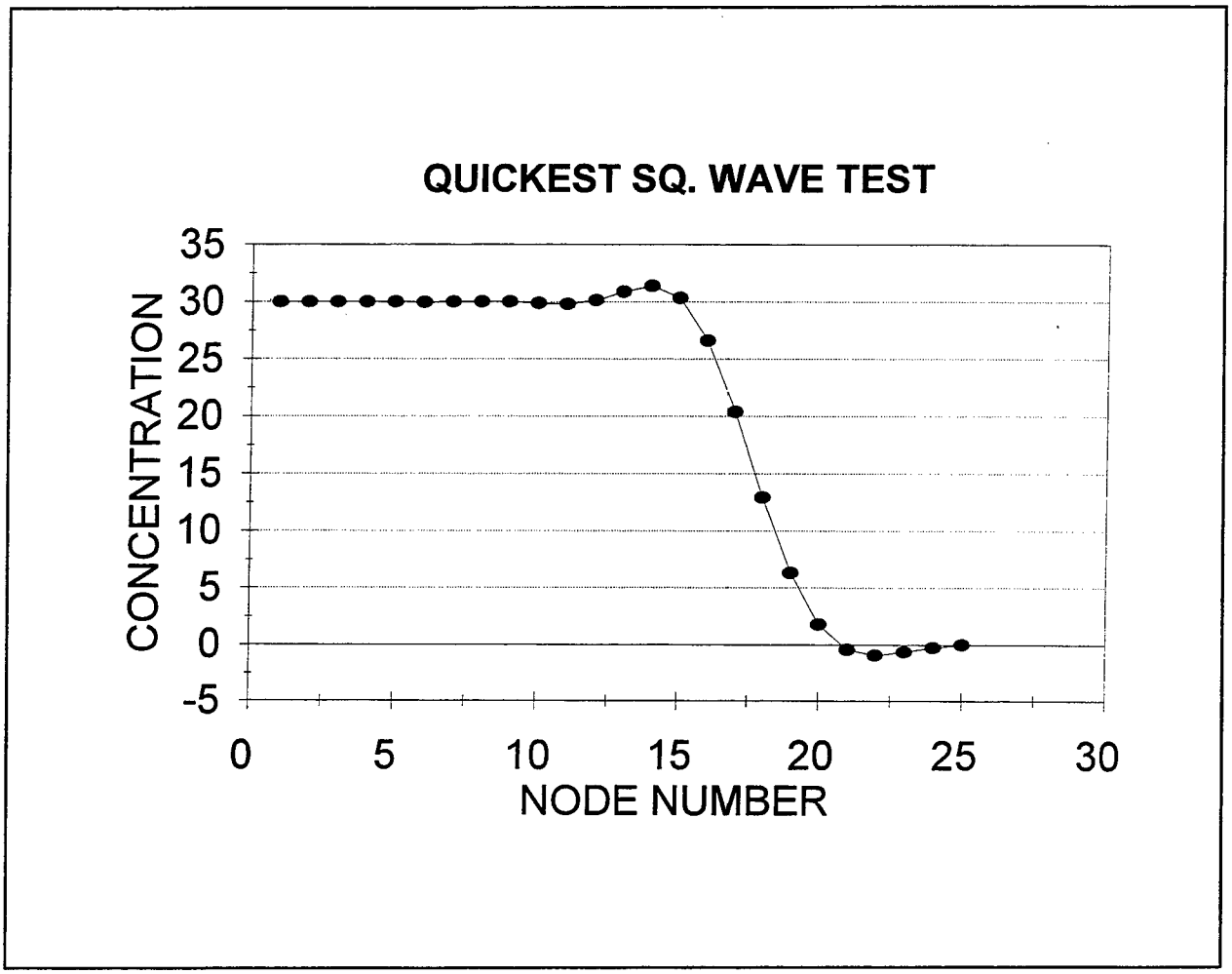


Figure 3. Square wave test result with structured grid QUICKEST multipliers

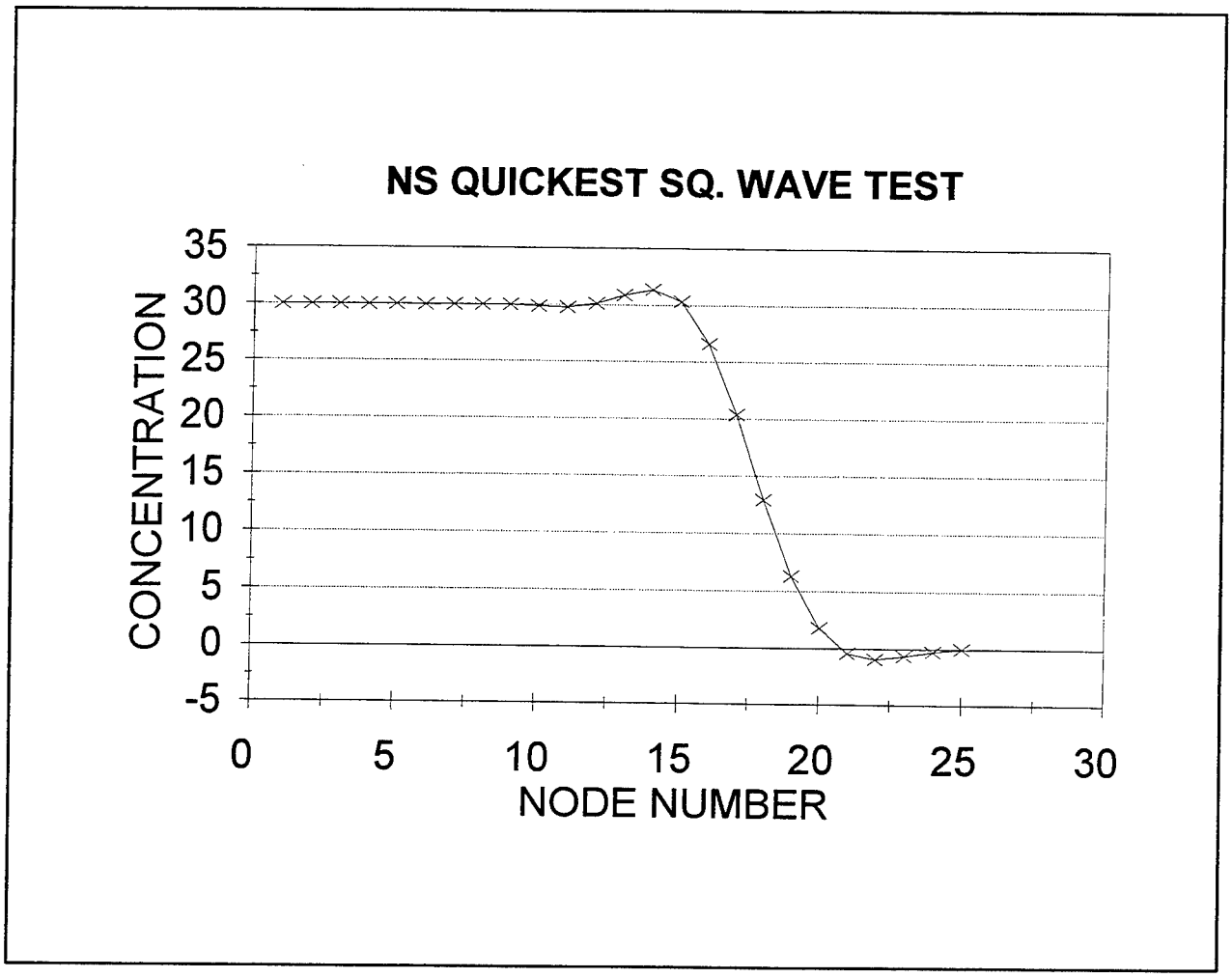


Figure 4. Square wave test result with unstructured grid QUICKEST multipliers

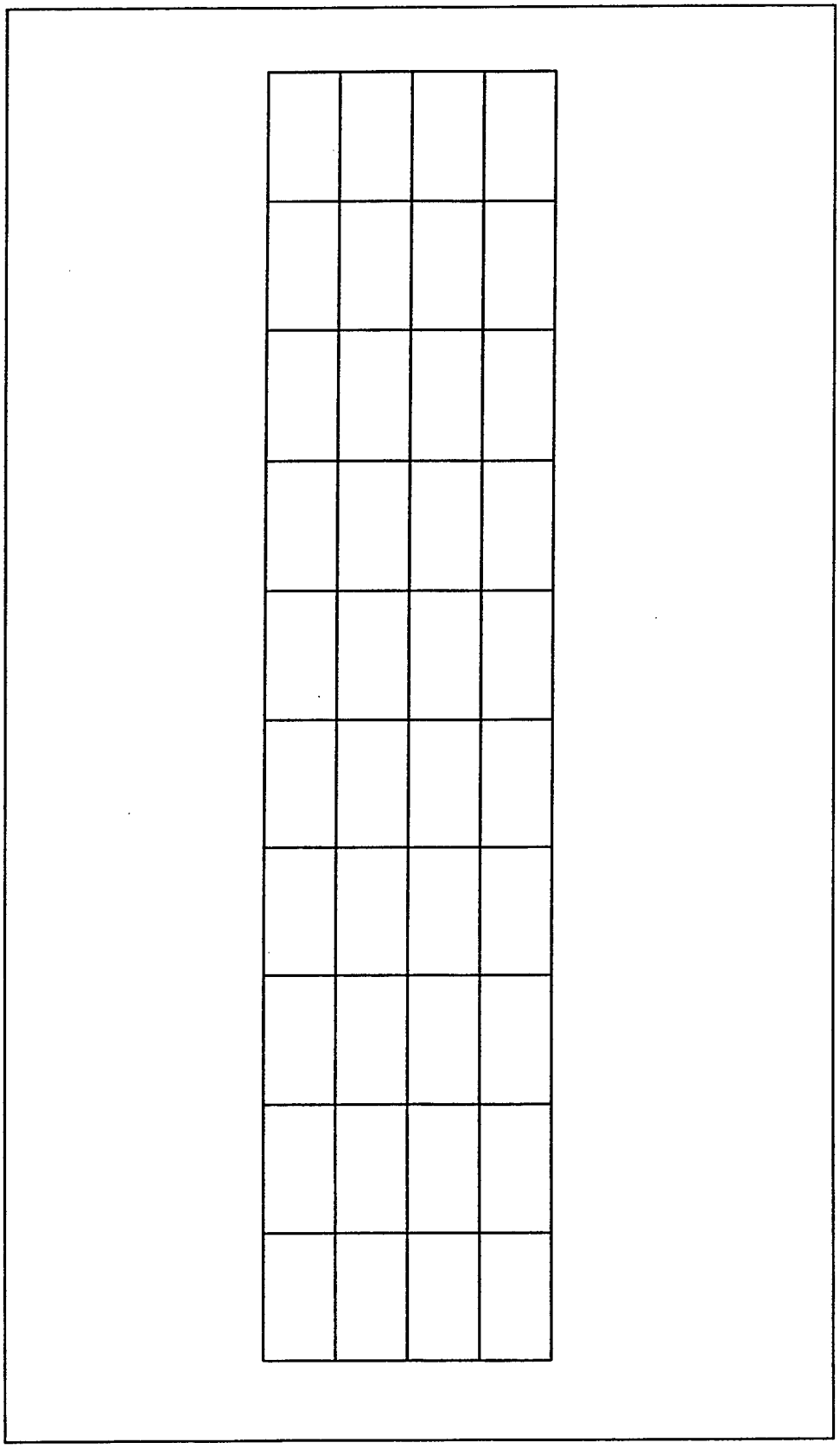


Figure 5. Transport test grid

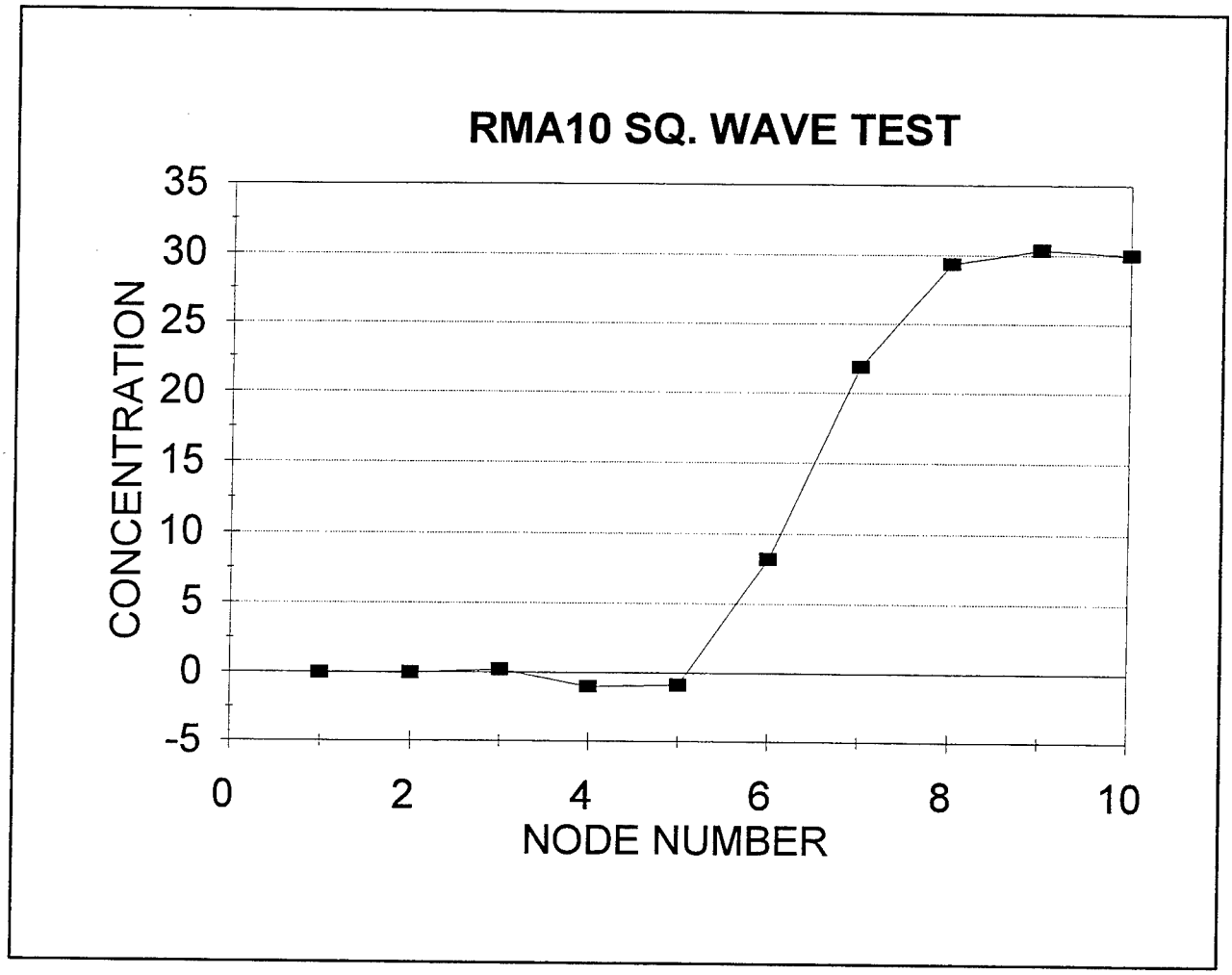


Figure 6. Square wave test result for RMA10-ICM transport test

4 Summary and Recommendations

A linkage methodology has been developed, implemented, and tested that allows CE-QUAL-ICM to use unstructured grid hydrodynamic information from the RMA10 finite element model. This has been accomplished through the development of two postprocessor programs, which provide flow and linkage information to ICM. Initial testing of volume conservation suggests that local volume conservation problems inherent within RMA10 limit the general skill of the linkage system; however, a procedure for correcting the volume errors within ICM is outlined. Irrespective, results of the present study suggest that further work towards eliminating volume conservation errors within RMA10 is warranted. In addition, a centroid-based QUICKEST advection scheme consistent with the unstructured form of RMA10 has been developed, implemented, and tested. Testing of the scheme shows that it is mass conservative and that it provides results consistent with the well-tested structured grid version of QUICKEST previously employed in ICM.

With respect to future refinements of the linkage methodology, in addition to either correcting for or eliminating the local volume conservation errors obtained from RMA10, the hydrodynamic processor FEMCONVT needs to be generalized to accommodate combinations of triangular and quadrilateral elements. Furthermore, additional work is required to integrate vertical eddy diffusivities within RMA10 so that cell face values can be provided to ICM. Subsequent to these and other more minor improvements outlined in Chapter 2, the complete linkage methodology should be tested using a real time-varying field application.

References

Brigham Young University. (1994). "TABS Primer," Engineering Computer Graphics Laboratory, Salt Lake City, UT.

Cerco, C. F., and Cole, T. (1995). "User's guide to the CE-QUAL-ICM three-dimensional eutrophication model, Release Version 1.0," Technical Report EL-95-15, U.S. Army Engineer Waterways Experiment Station, Vicksburg, MS.

Chapman, R. S. (1988). "Analysis of the numerical and physical mixing characteristics of the WASP box model," Ray Chapman and Associates, Final Report to the U.S. Army Engineer Waterways Experiment Station, Vicksburg, MS.

_____. (1992). "Research and development towards improving the predictive capability of CE-QUAL-W2," Ray Chapman and Associates, Final Report to the U.S. Army Engineer Waterways Experiment Station, Vicksburg, MS.

Chapman, R. S., and Cole T. M. (1992). "Improved thermal predictions in CE-QUAL-W2," *Proceedings, Water Forum '92, ASCE, Baltimore, MD.*

Dortch, M. S., Chapman, R. S., and Abt, S. R. (1991). "Application of three-dimensional residual transport," *Journal of Hydraulic Engineering*, Am. Soc. Civil Eng., 118(6), 831-848.

Johnson, B. H., Kim, K. W., Heath, R. E., and Butler, H. L. (1991). "Development and verification of a three-dimensional numerical hydrodynamic, salinity, and temperature model of Chesapeake Bay," Technical Report HL-91-7, U.S. Army Engineer Waterways Experiment Station, Vicksburg, MS.

Norton, W. R., King, I. P, and Orlob, G. T. (1973). "A finite element model for lower Granite Reservoir," Water Resource Engineers, Inc., Walnut Creek, CA.

Thomas, W. A., and McAnally, W. H., Jr. (1990). "User's manual for the generalized computer program system: Open-channel flow and sedimentation, TABS-2," Instruction Report HL-85-1, U.S. Army Engineer Waterways Experiment Station, Vicksburg MS.

Wang, P. F., Martin, J. L., Wool, T., and Dortch, M. S. (1996). "CE-QUAL-ICM/TOXI, a three-dimensional contaminant model for surface water: Model theory and user guide," Draft Technical Report, U.S. Army Engineer Waterways Experiment Station, Vicksburg, MS.

Appendix A
MAPPER Description and Source Code Listing

The following is a brief description of the internal steps followed by the program MAPPER.C that generates the MAP and GEO files needed by the ICM model. This program is run after the RMA10 program is executed. There is a program called R4ICR10.F written by the U.S. Army Engineer Waterways Experiment Station Coastal and Hydraulics Laboratory that extracts the two files used as input by MAPPER.C from the RMA10 binary output file. A listing of the MAPPER.C source code follows the description.

The step by step operations of MAPPER.C are listed below.

Read the node-to-element connection table. Allocates each "Element" as a column and assigns an ID.

Subroutine call: readElementConnectionFile()

Subroutine description: The routine opens and reads the element to node connection table file. This file provides information that uniquely relates an element or column identification number to the associated nodes. Additional information obtained from this file is the coordinates of each node. At this point, it is assumed that RMA10 node identification numbers are being used. This routine creates a unique "arc" for each side of the surface layer elements, which can be either trapezoidal or triangular.

Source of input file: Input file is generated by R4ICR10.f.

Read boundary information.

Subroutine call: readBoundaryArcFile()

Subroutine description: Routine reads a file that contains information about all "special" nodes in the RMA10 grid. These special nodes are nodes at which some physical parameters are specified such as head values, wall boundaries, open boundaries, interior boundaries, etc. This information, in addition to the node data from Step 1, completely describes the RMA10 grid structure.

Source of input file: Input file is generated by R4ICR10.f.

Relate each arc to the adjacent columns.

Subroutine call: mapColumnsToArcs()

Subroutine description: Routine determines the adjacent column identification numbers for each arc.

Relate each column to its neighbors.

Subroutine call: mapNeighborColumnsForEachColumn()

Subroutine description: Routine determines for each column, all the columns that share a common arc. Thus, a list is generated for each column that contains all of its neighbor columns.

Determine arc type.

Subroutine call: determineArcTypeF()

Subroutine description: Routine examines each arc and determines its type, which is defined to be (a) INTERNAL - arc that is internal to the grid, (b) BOUNDARY - arc lies at an open boundary, and (c) WALL - arc lies at the wall of the grid. The arc type is defined using the nodal boundary data.

Determine upstream and downstream columns.

Subroutine call: mapUpstreamAndDownstreamColumnsForEachColumnF()

Subroutine description: Routine defines adjacent columns.

Create a list of all elements in the 3-D grid.

Subroutine call: createElementList()

Subroutine description: Routine will generate a list of all elements in the RMA10 grid. First, it generates surface layer or column elements. Next, for

each column, it will define elements within that column from the surface down. The order of elements is that defined by RMA10.

Map neighboring elements.

Subroutine call: mapElementToNeighborElements()

Subroutine description: Routine determines neighboring elements. This is accomplished by using the column information previously defined.

Check for internal and boundary flow faces.

Subroutine call: generateHorizontalFlowFaceListF()

Subroutine description: Routine examines each arc, its type. If the type is "INTERNAL" or "BOUNDARY," it will create a flow-face for that arc; this is done in layer order starting at the surface then going down. For each defined flow face, the adjacent elements of that flow face are listed.

Determine far upstream elements for each horizontal flow face.

Subroutine call: mapHorizontalFlowFaces()

Subroutine description: Routine determines for each flow face its far upstream and far downstream element IDs using the adjacent element ID information.

Create the vertical flow face list.

Subroutine call: mapVerticalFlowFaces()

Subroutine description: Routine uses the column information defined above and creates a vertical flow face between each element. It also does the vertical mapping of upstream and downstream elements.

Define tags that relate RMA10 and ICM flow faces.

Subroutine call: setFlowFaceIdentifierTags()

Subroutine description: Routine examines each flow face and tags it with a unique identifier. This unique identifier is defined as (a) horizontally oriented flow face (the top center node ID of each flow face); and (b) vertically oriented flow face (the RMA10 element ID of the element below the flow face).

Number ICM flow faces.

Subroutine call: renumberAllFlowFacesForOutputF()

Subroutine description: Routine renumbers the flow face identifications consistent with the ICM convention.

Calculate surface areas and the centroid locations.

Subroutine call: calculateColumnSurfaceAreaAndCentroid()

Subroutine description: Routine will calculate surface area and centroid location using the surface node coordinates.

Calculate required distances between centroids.

Subroutine call: calculateDistanceBetweenCentroidsF()

Subroutine description: Routine calculates the following length scale information for each flow face:

(a) BIL = Distance between far upstream element centroid and upstream element centroid.

(b) BI = Distance between upstream element centroid and the downstream element centroid.

(c) BIR = Distance between downstream element centroid and far downstream element centroid.

(d) BID = Distance between flow face and the upstream element centroid.

Print "MAP FILE".

Subroutine call: printFlowFaceListF()

Subroutine description: Routine writes "MAP" file data.

Print "GEO FILE".

Subroutine call: printGeoFileF()

Subroutine description: Routine writes "GEO" file data.

```
/* PROGRAM NAME: MAPPER.c

  PROGRAMMER: Terry K. Gerald

  PURPOSE: Build the "MAP" file and the "GEO" file for ICM from an RMA10
       finite element model grid.
*/

#include <stdio.h>
#include <string.h>
#include <math.h>

#define  TRUE                  1
#define  FALSE                 0

#define  YES                   1
#define  NO                    0


#define  OK                          1


#define  SET                         1
#define  NOT_SET                     2


#define  GE                    10
#define  GNN                   20
#define  OTHERSTUFF                  30

#define  TRAPAZOID             100
#define  TRIANGLE              200

#define  NUMBER_OF_LAYERS            3
```

```
#define  MAXARGS                  20


#define  INTERNAL                 70
#define  BOUNDARY                    80
#define  WALL                           90


#define  VERTICAL                 100
#define  HORIZONTAL                 200
#define  INBETWEEN                  300

/* ____________________________________________________________ */



typedef struct ELEMENT_BLOCK_STRUCT
{ struct ELEMENT_BLOCK_STRUCT  *pLastElementBlockPtr;
 struct ELEMENT_BLOCK_STRUCT  *pNextElementBlockPtr;
 int   elementId;
 int   rma10ElementId;
 int   columnId;
 int   elementType;
 int   arcCount;
 int   layerId;
 int   faceId[6];
 int   commonNeighborElementId[4];
 int   commonNeighbor2ElementId[4];

 int   bottomFlowFaceId;
 struct ELEMENT_BLOCK_STRUCT *elementAbovePtr;
 struct ELEMENT_BLOCK_STRUCT *elementBelowPtr;
} ELEMENT_BLOCK;
```

```
typedef         ELEMENT_BLOCK *PELEMENT_BLOCK;

ELEMENT_BLOCK     *pHeadElementListBlockPtr;

int             globalElementCount = 0;

PELEMENT_BLOCK    *lookupElementPtrArray = NULL;


typedef struct COLUMN_BLOCK_STRUCT
{
 struct COLUMN_BLOCK_STRUCT  *pLastColumnBlockPtr;
 struct COLUMN_BLOCK_STRUCT  *pNextColumnBlockPtr;
 int   columnId;
 int   columnType;
 int   arcCount;
 int   numberOfLayers;
 int   arcIds[4];
 int   commonNeighborColumnCount;
 int   commonNeighborColumnId[4];
 int   commonNeighborArcId[4];
 int   commonNeighbor2ColumnId[4];

 float  surfaceArea;
 float  centroidX, centroidY;

 PELEMENT_BLOCK elementPtr;
 PELEMENT_BLOCK surfaceElementPtr;

} COLUMN_BLOCK;

typedef         COLUMN_BLOCK *PCOLUMN_BLOCK;

COLUMN_BLOCK      *pHeadColumnListBlockPtr;
```

```
int             globalColumnCount = 0;

PCOLUMN_BLOCK     *lookupColumnPtrArray = NULL;


typedef struct NODE_BLOCK_STRUCT
{ struct NODE_BLOCK_STRUCT  *lastNodeBlockPtr;
 struct NODE_BLOCK_STRUCT  *nextNodeBlockPtr;
 int   nodeId;
 float  coordinates[3];
} NODE_BLOCK;

typedef NODE_BLOCK *PNODE_BLOCK;

PNODE_BLOCK       pHeadNodeBlockPtr = NULL;

int             globalNodeCount = 0;

PNODE_BLOCK       *lookupNodePtrArray = NULL;


typedef struct ARC_BLOCK_STRUCT
{ struct ARC_BLOCK_STRUCT  *pLastArcBlockPtr;
 struct ARC_BLOCK_STRUCT  *pNextArcBlockPtr;
 int   arcId;
 int   nodeId[3];
 int   adjacentColumn[2];
 int   type;
} ARC_BLOCK;

typedef ARC_BLOCK  *PARC_BLOCK;

PARC_BLOCK        pHeadArcListBlockPtr = NULL;
```

```
int             globalArcCount = 0;


typedef struct FLOWFACE_BLOCK_STRUCT
{ struct FLOWFACE_BLOCK_STRUCT  *pLastFlowFaceBlockPtr;
 struct FLOWFACE_BLOCK_STRUCT  *pNextFlowFaceBlockPtr;
 struct ARC_BLOCK_STRUCT  *arcPtr;
 int   identifierTag;
 int   arcId;
 int   flowFaceId;
 int   modelFlowFaceId;
 int   layerId;
 int   farDownstreamElementId;
 int   downstreamElementId;
 int   upstreamElementId;
 int   farUpstreamElementId;
 int   type;
 int   orientation;
 int   adjacentColumn[2];
 int   adjacentElement[2];
 float  distanceBetweenCentroids;
 float  upstreamDistanceBetweenCentroidAndFace;
 float  downstreamDistanceBetweenCentroidAndFace;
 float  distanceBetweenUpstreamCentroids;
 float  distanceBetweenDownstreamCentroids;

} FLOWFACE_BLOCK;

typedef FLOWFACE_BLOCK  *PFLOWFACE_BLOCK;

PFLOWFACE_BLOCK      pHeadFlowFaceListBlockPtr = NULL;
PFLOWFACE_BLOCK      pHeadSurfaceFlowFaceListBlockPtr = NULL;
PFLOWFACE_BLOCK      pTailSurfaceFlowFaceListBlockPtr = NULL;
```

```
int             globalFlowFaceCount = 0;

PFLOWFACE_BLOCK     *lookupFlowFacePtrArray = NULL;


typedef struct BOUNDARY_NODE_BLOCK_STRUCT
{ struct BOUNDARY_NODE_BLOCK_STRUCT  *lastBoundaryNodeBlockPtr;
 struct BOUNDARY_NODE_BLOCK_STRUCT  *nextBoundaryNodeBlockPtr;
 int   boundaryNodeId;
} BOUNDARY_NODE_BLOCK;

typedef BOUNDARY_NODE_BLOCK *PBOUNDARY_NODE_BLOCK;

PBOUNDARY_NODE_BLOCK     pHeadBoundaryNodeBlockPtr = NULL;

/* int          globalBoundaryNodeCount = 0;  */


typedef struct SURFACE_NODE_LIST_STRUCT
{
 int ndep;
 int nref;
} SURFACE_NODE_LIST;

typedef SURFACE_NODE_LIST *PSURFACE_NODE_LIST;

PSURFACE_NODE_LIST  surfaceNodeList;


/* ____________________________________________________________________*/

                        /* function prototypes          */
```

```
NODE_BLOCK
 *allocateNewNodeBlockF();


BOUNDARY_NODE_BLOCK
 *allocateNewBoundaryNodeBlockF();


ARC_BLOCK
 *allocateNewSegmentArcBlockF();


ELEMENT_BLOCK
 *allocateNewElementBlockF();


COLUMN_BLOCK
 *allocateNewColumnBlockF();


FLOWFACE_BLOCK
 *allocateNewFlowFaceBlockF();


PNODE_BLOCK
 findNodePtrF(int);


PARC_BLOCK
 findArcPtrF(int);


PCOLUMN_BLOCK
 lookupColumnPtrF(int);
```

```
PELEMENT_BLOCK
 lookupElementPtrF(int);


PFLOWFACE_BLOCK
 lookupFlowFacePtrF(int);


int
 iCheckIfCommonNodeInTwoArcsF(PARC_BLOCK, PARC_BLOCK);


void
 readBoundaryArcFile();

int
 getColumn(FILE *, char *, char *);

int
 searchForNFIX(FILE *);


void
 allocateNewElementByColumnBlockF( COLUMN_BLOCK *, int );


/* ____________________________________________________________ */

/* ____________________________________________________________ */

                        /* MAIN ROUTINE                */

void main()
```

```
{
                                    /* Read the node-to-element connection
                                       table. Allocates each "Element" as
                                       a column. Column ID is also read in. */
readElementConnectionFile();

                                    /* Read from an RMA10 output file, which
                                    nodes are BOUNDARY nodes          */
readBoundaryArcFile();
                                    /* Releate to each arc in a column, the
                                       two columns on each side          */
mapColumnsToArcs();

                                    /* Releate each column to its neighbors */
mapNeighborColumnsForEachColumn();

                                    /* Determine for each arc if it is an
                                       1) INTERNAL arc, 2) BOUNDARY arc or a
                                       3) WALL arc!                      */
determineArcTypeF();

                                    /* Map upstream and downstream columns  */

mapUpstreamAndDownstreamColumnsForEachColumnF();

                                    /* Create all elements               */
createElementList();

                                    /* Map for each element its neighbor
                                       elements                          */
mapElementToNeighborElements();

                                    /* Generate each horizontal flow-face
                                       utilizing the arc list            */
```

```
generateHorizontalFlowFaceListF();

                                /* Determine relationship between the
                                   flow-face and it's adjacent elements */
mapHorizontalFlowFaces();
                                /* From the column lists generate the
                                   vertical flow-faces and determine
                                   their adjacent element ids        */
mapVerticalFlowFaces();

                                /* Need to reference each flow-face
                                   with some type of information that
                                   RMA10 is aware of so the flow-faces
                                   can be ordered in way that perserves
                                   their geometric relationships to
                                   their elements .                  */
setFlowFaceIdentifierTags();

                                /* Renumber the flow-faces so that they
                                   are numbered increasing by layer
                                   instead of by column              */
renumberAllFlowFacesForOutputF();

                                /* Calculate the geometry for each
                                   element                           */
calculateColumnSurfaceAreaAndCentroid();

calculateDistanceBetweenCentroidsF();


                                /* Print the "MAP" file              */
printFlowFaceListF();
                                /* Print the "GEO" file              */
```

```
printGeoFileF();

}


/* ____________________________________________________________ */


NODE_BLOCK *allocateNewNodeBlockF()
{

int i;
NODE_BLOCK *aNodeBlock;

static
PNODE_BLOCK lastNodePtr;


aNodeBlock = (PNODE_BLOCK)calloc(1,sizeof(NODE_BLOCK) );

aNodeBlock->lastNodeBlockPtr = NULL;
aNodeBlock->nextNodeBlockPtr = NULL;

aNodeBlock->nodeId = ++globalNodeCount;


                        /* Set list ptrs          */

if(pHeadNodeBlockPtr == NULL)
 { pHeadNodeBlockPtr = aNodeBlock;
 }
```

```
else
 { lastNodePtr->nextNodeBlockPtr = aNodeBlock;
  aNodeBlock->lastNodeBlockPtr = lastNodePtr;
 }

lastNodePtr = aNodeBlock;


return(aNodeBlock);
}


/* ____________________________________________________________ */


BOUNDARY_NODE_BLOCK *allocateNewBoundaryNodeBlockF()
{

int i;
BOUNDARY_NODE_BLOCK *aBoundaryNodeBlockPtr;

static
PBOUNDARY_NODE_BLOCK lastBoundaryNodePtr;


aBoundaryNodeBlockPtr = (PBOUNDARY_NODE_BLOCK)calloc(1,sizeof(BOUNDARY_NODE_BLOCK) );

aBoundaryNodeBlockPtr->lastBoundaryNodeBlockPtr = NULL;
aBoundaryNodeBlockPtr->nextBoundaryNodeBlockPtr = NULL;
```

```
                                    /* Set list ptrs                */

if(pHeadBoundaryNodeBlockPtr == NULL)
 { pHeadBoundaryNodeBlockPtr = aBoundaryNodeBlockPtr;
 }
else
 { lastBoundaryNodePtr->nextBoundaryNodeBlockPtr = aBoundaryNodeBlockPtr;
   aBoundaryNodeBlockPtr->lastBoundaryNodeBlockPtr = lastBoundaryNodePtr;
 }

lastBoundaryNodePtr = aBoundaryNodeBlockPtr;


return(aBoundaryNodeBlockPtr);
}


/* ____________________________________________________________________ */


ARC_BLOCK *allocateNewArcBlockF()
{

 int i;
 ARC_BLOCK *aArcBlock;

 static
 PARC_BLOCK lastArcPtr;


 aArcBlock = (PARC_BLOCK)calloc(1,sizeof(ARC_BLOCK) );

 aArcBlock->pLastArcBlockPtr = NULL;
 aArcBlock->pNextArcBlockPtr = NULL;
```

```
  aArcBlock->arcId = ++globalArcCount;

  aArcBlock->adjacentColumn[0] = -1;
  aArcBlock->adjacentColumn[1] = -1;

                                    /* Set list ptrs                */

  if(pHeadArcListBlockPtr == NULL)
   { pHeadArcListBlockPtr = aArcBlock;
   }
  else
   { lastArcPtr->pNextArcBlockPtr = aArcBlock;
     aArcBlock->pLastArcBlockPtr = lastArcPtr;
   }

  lastArcPtr = aArcBlock;


  return(aArcBlock);
 }



 /* ____________________________________________________________ */


 ELEMENT_BLOCK *allocateNewElementBlockF()
 {
  int i;
  ELEMENT_BLOCK *anElementBlockPtr;

  static
  ELEMENT_BLOCK *lastElementPtr;
```

```
anElementBlockPtr = (PELEMENT_BLOCK)calloc(1,sizeof(ELEMENT_BLOCK) );

anElementBlockPtr->pLastElementBlockPtr = NULL;
anElementBlockPtr->pNextElementBlockPtr = NULL;

anElementBlockPtr->elementAbovePtr=NULL;
anElementBlockPtr->elementBelowPtr=NULL;

for(i=0; i<6; i++)
  anElementBlockPtr->faceId[i] = -1;


globalElementCount++;

anElementBlockPtr->elementId= globalElementCount;


                                        /* Set list ptrs                */

if(pHeadElementListBlockPtr == NULL)
 { pHeadElementListBlockPtr = anElementBlockPtr;
   lastElementPtr = anElementBlockPtr;
 }
else
 { lastElementPtr->pNextElementBlockPtr = anElementBlockPtr;
   anElementBlockPtr->pLastElementBlockPtr = lastElementPtr;
   lastElementPtr = anElementBlockPtr;
 }



return(anElementBlockPtr);

}
```

```
/* ____________________________________________________________ */

COLUMN_BLOCK *allocateNewColumnBlockF()
{

 int i;
 COLUMN_BLOCK *aColumnBlockPtr;

 static
 PCOLUMN_BLOCK lastColumnPtr;


 aColumnBlockPtr = (PCOLUMN_BLOCK)calloc(1,sizeof(COLUMN_BLOCK) );

 aColumnBlockPtr->pLastColumnBlockPtr = NULL;
 aColumnBlockPtr->pNextColumnBlockPtr = NULL;

 aColumnBlockPtr->columnId = ++globalColumnCount;


 aColumnBlockPtr->commonNeighborColumnCount = 0;

 for(i=0; i<4; i++)
  {  aColumnBlockPtr->commonNeighborColumnId[i] = -1;
     aColumnBlockPtr->commonNeighborArcId[i] = -1;
     aColumnBlockPtr->commonNeighbor2ColumnId[i] = -1;
  }


                              /* Set list ptrs              */
```

```
if(pHeadColumnListBlockPtr == NULL)
 { pHeadColumnListBlockPtr = aColumnBlockPtr;
 }
else
 { lastColumnPtr->pNextColumnBlockPtr = aColumnBlockPtr;
   aColumnBlockPtr->pLastColumnBlockPtr = lastColumnPtr;
 }

lastColumnPtr = aColumnBlockPtr;

return(aColumnBlockPtr);

}


/* ____________________________________________________________ */


FLOWFACE_BLOCK *allocateNewFlowFaceBlockF()
{

 int i;
 FLOWFACE_BLOCK *aFlowFaceBlockPtr;

 static
 PFLOWFACE_BLOCK lastFlowFacePtr;


 aFlowFaceBlockPtr = (PFLOWFACE_BLOCK)calloc(1,sizeof(FLOWFACE_BLOCK) );

 aFlowFaceBlockPtr->pLastFlowFaceBlockPtr = NULL;
 aFlowFaceBlockPtr->pNextFlowFaceBlockPtr = NULL;
 aFlowFaceBlockPtr->arcPtr = NULL;
```

```
aFlowFaceBlockPtr->flowFaceId = ++globalFlowFaceCount;

aFlowFaceBlockPtr->adjacentColumn[0] = -1;
aFlowFaceBlockPtr->adjacentColumn[1] = -1;

aFlowFaceBlockPtr->adjacentElement[0] = -1;
aFlowFaceBlockPtr->adjacentElement[1] = -1;


                                    /* Set list ptrs                */

if(pHeadFlowFaceListBlockPtr == NULL)
 { pHeadFlowFaceListBlockPtr = aFlowFaceBlockPtr;
 }
else
 { lastFlowFacePtr->pNextFlowFaceBlockPtr = aFlowFaceBlockPtr;
   aFlowFaceBlockPtr->pLastFlowFaceBlockPtr = lastFlowFacePtr;
 }

lastFlowFacePtr = aFlowFaceBlockPtr;


return(aFlowFaceBlockPtr);


}


/* ____________________________________________________________________ */

void calculateCentroidOfGeneralizedTriangle(x,y,cList)
    float x[3], y[3];
    float *cList;
```

```
{
 float A, d, e;
 float x1,y1, x2,y2, x3,y3;
 float z1, z2, z3;

                                        /* First calculate arc midpoint
                                           locations                    */
 x1 = (x[1]+x[2]) / 2.0;
 y1 = (y[1]+y[2]) / 2.0;

 x2 = (x[2]+x[3]) / 2.0;
 y2 = (y[2]+y[3]) / 2.0;

 x3 = (x[3]+x[1]) / 2.0;
 y3 = (y[3]+y[1]) / 2.0;


 A = (x2*y3 - x3*y2) -(x1*y3 - x3*y1) + (x1*y2 - x2*y1);
 z1 = x1*x1 + y1*y1;
 z1 = x2*x2 + y2*y2;
 z1 = x3*x3 + y3*y3;


 d = (z1*y3 - z1*y2 - z2*y3 + z2*y1 + z3*y2 - z3*y1) /( -2.0*A);

 e = (-z1*x3 + z1*x2 + z2*x3 - z2*x1 - z3*x2 + z3*x1) /( -2.0*A);

 cList[0] = d; cList[1] = e;

}

/* ____________________________________________________________ */

void calculateColumnSurfaceAreaAndCentroid()
```

```
{

PCOLUMN_BLOCK  columnPtr;
PARC_BLOCK     arcPtr;
PNODE_BLOCK    nodePtr;

int         arcId;
int         i, j, iTemp;
int         nodeIds[4];
int         nodeCount;
float       x[5], y[5];
float       area;
int         lastCornerNode;
float       centroidX, centroidY;
float       cList[2];

int         foundIt;

int         nodeIdList[13], nodeIdListCount;


columnPtr = pHeadColumnListBlockPtr;

while(columnPtr != NULL)
 {
  nodeCount = 0;
                                        /* Each column has 4 potential arcs   */


  nodeIdListCount = 0;

  for(i=0; i<4; i++)
```

```
 {
  if(columnPtr->arcIds[i] < 1) continue;
  arcId = columnPtr->arcIds[i];
  arcPtr = findArcPtrF(arcId);

                                        /* Check if arc already encounterd   */
  foundIt = NO;
  for(j=0; j<nodeIdListCount; j++)
   if( arcPtr->nodeId[0] == nodeIdList[j])
    foundIt = YES;

  if(foundIt == NO)
    nodeIdList[nodeIdListCount++] = arcPtr->nodeId[0];


  foundIt = NO;
  for(j=0; j<nodeIdListCount; j++)
   if( arcPtr->nodeId[2] == nodeIdList[j])
    foundIt = YES;

  if(foundIt == NO)
    nodeIdList[nodeIdListCount++] = arcPtr->nodeId[2];

 }

nodeCount = nodeIdListCount;
for(i=0; i<nodeIdListCount; i++)
 nodeIds[i] = nodeIdList[i];


                                        /* Now get coors of these nodes      */
for(i=1; i<=nodeCount; i++)
 { nodePtr = findNodePtrF(nodeIds[i-1]);
```

```
    x[i] = nodePtr->coordinates[0];
    y[i] = nodePtr->coordinates[1];
  }

                                        /* Check if type of element        */
switch(nodeCount)

{ case 3:
      area = 0.5*(  x[1]*y[2] + x[2]*y[3] + x[3]*y[1]
                -y[1]*x[2] - y[2]*x[3] - y[3]*x[1] );

      calculateCentroidOfGeneralizedTriangle(x,y,cList);
      centroidX = cList[0];
      centroidY = cList[1];

      break;

  case 4:
      area = 0.5*(  x[1]*y[2] + x[2]*y[3] + x[3]*y[4] + x[4]*y[1]
                -y[1]*x[2] - y[2]*x[3] - y[3]*x[4] - y[4]*x[1]);

      centroidX = (x[1] + x[2] + x[3] + x[4])/4.0;
      centroidY = (y[2] + y[3] + y[4] + y[1])/4.0;

      break;
}

columnPtr->surfaceArea = area;

columnPtr->centroidX = centroidX;
columnPtr->centroidY = centroidY;
```

```
  columnPtr = columnPtr->pNextColumnBlockPtr;

 }

}

/* ____________________________________________________________ */

float
 calculate_BI_F(PFLOWFACE_BLOCK  flowFacePtr)

{

 PELEMENT_BLOCK   downstreamElementPtr, upstreamElementPtr;
 int          downstreamElementId, upstreamElementId;

 PCOLUMN_BLOCK    upstreamColumnPtr, downstreamColumnPtr;
 int          downstreamColumnId,upstreamColumnId;

 float         x,y, x1,y1, x2,y2;
 double          d2;
 float         d;


 upstreamElementId = flowFacePtr->upstreamElementId;
 upstreamElementPtr = lookupElementPtrF(upstreamElementId);
 if(upstreamElementPtr != NULL)
  {
    upstreamColumnId = upstreamElementPtr->columnId;
    upstreamColumnPtr = lookupColumnPtrF(upstreamColumnId);
  }
```

```
else
 upstreamColumnPtr = NULL;


downstreamElementId = flowFacePtr->downstreamElementId;
downstreamElementPtr = lookupElementPtrF(downstreamElementId);
if(downstreamElementPtr != NULL)
 {
  downstreamColumnId = downstreamElementPtr->columnId;
  downstreamColumnPtr = lookupColumnPtrF(downstreamColumnId);
 }
else
 downstreamColumnPtr = NULL;


if( (upstreamColumnPtr != NULL) && (downstreamColumnPtr != NULL) )
 {
   x1 = upstreamColumnPtr->centroidX;
   y1 = upstreamColumnPtr->centroidY;

   x2 = downstreamColumnPtr->centroidX;
   y2 = downstreamColumnPtr->centroidY;

   d2 = (x1-x2)*(x1-x2) + (y1-y2)*(y1-y2);
   d2 = sqrt(d2);

   d = d2;
 }
else
 d = 0.0;

 return(d);
}
```

```
/* ____________________________________________________________ */

float
calculate_BID_F( PFLOWFACE_BLOCK  flowFacePtr )

{

PELEMENT_BLOCK   downstreamElementPtr, upstreamElementPtr;
int          downstreamElementId, upstreamElementId;

PELEMENT_BLOCK   farDownstreamElementPtr, farUpstreamElementPtr;
int          farDownstreamElementId, farUpstreamElementId;

PCOLUMN_BLOCK    downstreamColumnPtr, upstreamColumnPtr;
int          downstreamColumnId,upstreamColumnId;

PCOLUMN_BLOCK    farDownstreamColumnPtr, farUpstreamColumnPtr;
int          farDownstreamColumnId, farUpstreamColumnId;


float         x,y, x1,y1, x2,y2, xA1,yA1, xA2,yA2;
PARC_BLOCK      arcPtr;

int          lineType;

PNODE_BLOCK     node1Ptr, node2Ptr;

int          node1, node2;

float         mCentroid, yIntCentroid;
float         m1, m2, b1, b2, d;
float         xBar, yBar;
double          d2;
float         TOLERANCE = 0.001;
```

```
d = 0.0;

upstreamElementId = flowFacePtr->upstreamElementId;
upstreamElementPtr = lookupElementPtrF(upstreamElementId);
if(upstreamElementPtr != NULL)
 {
   upstreamColumnId = upstreamElementPtr->columnId;
   upstreamColumnPtr = lookupColumnPtrF(upstreamColumnId);
 }
else
 upstreamColumnPtr = NULL;


downstreamElementId = flowFacePtr->downstreamElementId;
downstreamElementPtr = lookupElementPtrF(downstreamElementId);
if(downstreamElementPtr != NULL)
 {
   downstreamColumnId = downstreamElementPtr->columnId;
   downstreamColumnPtr = lookupColumnPtrF(downstreamColumnId);
 }
else
 downstreamColumnPtr = NULL;


if( (flowFacePtr != NULL) && (upstreamColumnPtr == NULL) )
 {

   arcPtr = flowFacePtr->arcPtr;
   node1 = arcPtr->nodeId[0];
   node2 = arcPtr->nodeId[2];

   node1Ptr = findNodePtrF(node1);
   xA1 = node1Ptr->coordinates[0];
   yA1 = node1Ptr->coordinates[1];
```

```
    node2Ptr = findNodePtrF(node2);
    xA2 = node2Ptr->coordinates[0];
    yA2 = node2Ptr->coordinates[1];
    x1 = (xA1 + xA2) / 2.0;
    y1 = (yA1 + yA2) / 2.0;

    x2 = downstreamColumnPtr->centroidX;
    y2 = downstreamColumnPtr->centroidY;

    d2 = (x1-x2)*(x1-x2) + (y1-y2)*(y1-y2);
    d2 = sqrt(d2);

    d = d2;

    return(d);
  }


if( (flowFacePtr != NULL) && (downstreamColumnPtr != NULL) )
 {

    x1 = upstreamColumnPtr->centroidX;
    y1 = upstreamColumnPtr->centroidY;

    x2 = downstreamColumnPtr->centroidX;
    y2 = downstreamColumnPtr->centroidY;


                                    /* Set line type                */
    if( (y2-y1) < TOLERANCE )
      lineType = HORIZONTAL;
    else
     if( (x2-x1) < TOLERANCE )
       lineType = VERTICAL;
```

```
    else
      lineType = INBETWEEN;


    switch(lineType)
     {  case HORIZONTAL:

            arcPtr = flowFacePtr->arcPtr;
            node1 = arcPtr->nodeId[0];
            node2 = arcPtr->nodeId[2];

            node1Ptr = findNodePtrF(node1);
            xA1 = node1Ptr->coordinates[0];
            yA1 = node1Ptr->coordinates[1];

            node2Ptr = findNodePtrF(node2);
            xA2 = node2Ptr->coordinates[0];
            yA2 = node2Ptr->coordinates[1];

            xBar = (xA1 + xA2) / 2.0;
            yBar = (yA1 + yA2) / 2.0;

            d2 =  xBar - x2;
            if( d2 < 0.0) d2 = -d2;

            d = d2;

            break;

       case VERTICAL:

            arcPtr = flowFacePtr->arcPtr;
            node1 = arcPtr->nodeId[0];
            node2 = arcPtr->nodeId[2];
```

```
        node1Ptr = findNodePtrF(node1);
        xA1 = node1Ptr->coordinates[0];
        yA1 = node1Ptr->coordinates[1];

        node2Ptr = findNodePtrF(node2);
        xA2 = node2Ptr->coordinates[0];
        yA2 = node2Ptr->coordinates[1];

        xBar = (xA1 + xA2) / 2.0;
        yBar = (yA1 + yA2) / 2.0;

        d2 =  yBar - y2;
        if( d2 < 0.0) d2 = -d2;

        d = d2;

        break;

    case INBETWEEN:

        m1 = (x2-x1) / (y2-y1);
        b1 = y1-m1*x1;

                                    /* Calculate equation of arc        */
        arcPtr = flowFacePtr->arcPtr;
        node1 = arcPtr->nodeId[0];
        node2 = arcPtr->nodeId[2];

        node1Ptr = findNodePtrF(node1);
        xA1 = node1Ptr->coordinates[0];
        yA1 = node1Ptr->coordinates[1];
```

```
        node2Ptr = findNodePtrF(node2);
        xA2 = node2Ptr->coordinates[0];
        yA2 = node2Ptr->coordinates[1];

        m2 = (xA2-xA1) / (yA2-yA1);
        b2 = yA1-m2*xA1;

                                    /* Calculate intersection location of
                                       the two lines                    */
        x = (b2-b1) / (m1-m2);
        y = (m1*b2 - m2*b1) / (m1-m2);


        d2 = (x-x2)*(x-x2) + (y-y2)*(y-y2);
        d2 = sqrt(d2);

        d = d2;

        break;
    }
  }


 return(d);

}

/* ____________________________________________________________ */

float
 calculate_BIL_F( PFLOWFACE_BLOCK  flowFacePtr )
```

```
{

PELEMENT_BLOCK   upstreamElementPtr;
int              upstreamElementId;

PCOLUMN_BLOCK    upstreamColumnPtr;
int              upstreamColumnId;

PELEMENT_BLOCK   farUpstreamElementPtr;
int              farUpstreamElementId;

PCOLUMN_BLOCK    farUpstreamColumnPtr;
int              farUpstreamColumnId;

float            x1,y1, x2,y2;

float            d;
double           d2;


upstreamElementId = flowFacePtr->upstreamElementId;
upstreamElementPtr = lookupElementPtrF(upstreamElementId);
if(upstreamElementPtr != NULL)
 {
   upstreamColumnId = upstreamElementPtr->columnId;
   upstreamColumnPtr = lookupColumnPtrF(upstreamColumnId);
 }
else
 upstreamColumnPtr = NULL;


farUpstreamElementId = flowFacePtr->farUpstreamElementId;
farUpstreamElementPtr = lookupElementPtrF(farUpstreamElementId);
if( (upstreamColumnPtr != NULL) && (farUpstreamElementPtr != NULL) )
```

```
    {
      farUpstreamColumnId = farUpstreamElementPtr->columnId;
      farUpstreamColumnPtr = lookupColumnPtrF(farUpstreamColumnId);

      x1 = upstreamColumnPtr->centroidX;
      y1 = upstreamColumnPtr->centroidY;

      x2 = farUpstreamColumnPtr->centroidX;
      y2 = farUpstreamColumnPtr->centroidY;

      d2 = (x1-x2)*(x1-x2) + (y1-y2)*(y1-y2);
      d2 = sqrt(d2);
      d = d2;

    }
   else
     d = 0.0;

 return(d);

}

/* ____________________________________________________________ */

float
 calculate_BIR_F( PFLOWFACE_BLOCK  flowFacePtr )

{

 PELEMENT_BLOCK   downstreamElementPtr;
 int              downstreamElementId;

 PCOLUMN_BLOCK    downstreamColumnPtr;
 int              downstreamColumnId;
```

```
PELEMENT_BLOCK   farDownstreamElementPtr;
int              farDownstreamElementId;

PCOLUMN_BLOCK    farDownstreamColumnPtr;
int              farDownstreamColumnId;

float            x1,y1, x2,y2;

float            d;
double           d2;


downstreamElementId = flowFacePtr->downstreamElementId;
downstreamElementPtr = lookupElementPtrF(downstreamElementId);
if(downstreamElementPtr != NULL)
 {
   downstreamColumnId = downstreamElementPtr->columnId;
   downstreamColumnPtr = lookupColumnPtrF(downstreamColumnId);
 }
else
 downstreamColumnPtr = NULL;


farDownstreamElementId = flowFacePtr->farDownstreamElementId;
farDownstreamElementPtr = lookupElementPtrF(farDownstreamElementId);
if( (downstreamColumnPtr != NULL) && (farDownstreamElementPtr != NULL) )
  {

    farDownstreamColumnId = farDownstreamElementPtr->columnId;
    farDownstreamColumnPtr = lookupColumnPtrF(farDownstreamColumnId);

    x1 = downstreamColumnPtr->centroidX;
    y1 = downstreamColumnPtr->centroidY;
```

```
      x2 = farDownstreamColumnPtr->centroidX;
      y2 = farDownstreamColumnPtr->centroidY;

      d2 = (x1-x2)*(x1-x2) + (y1-y2)*(y1-y2);
      d2 = sqrt(d2);

      d = d2;
    }
  else
    d = 0.0;


 return(d);

}

/* ____________________________________________________________ */


void calculateDistanceBetweenCentroidsF()

{

 FLOWFACE_BLOCK   *flowFacePtr;


                                    /* Examine each cell face and calculate
                                       the required distances          */


 flowFacePtr = pHeadFlowFaceListBlockPtr;

 while( (flowFacePtr != NULL) && (flowFacePtr->orientation == HORIZONTAL) )
```

```
{

 if( (flowFacePtr->type ==  INTERNAL) || (flowFacePtr->type ==  BOUNDARY) )
  {

                        /* BIL = Distance between farUpstreamElement
                               centroid and upstreamElement centroid.

                          BI =  Distance between upstreamElement
                               centroid and the downstreamElement
                               centroid.

                          BIR = Distance between downstreamElement
                               centroid and farDownstreamElement
                               centroid.

                          BID = Distance between flowface and the
                               downstreamElement.                  */


                        /* Calculate BI!                           */

    flowFacePtr->distanceBetweenCentroids =
      calculate_BI_F(flowFacePtr);


                        /* Calculate BID!                          */

    flowFacePtr->downstreamDistanceBetweenCentroidAndFace =
      calculate_BID_F(flowFacePtr);


                        /* Calculate BIL!                          */
```

```
      flowFacePtr->distanceBetweenUpstreamCentroids =
       calculate_BIL_F(flowFacePtr);

                                  /* Calculate BIR!                    */

      flowFacePtr->distanceBetweenDownstreamCentroids=
       calculate_BIR_F(flowFacePtr);

    }
                                  /* Go to next flow face              */

   flowFacePtr = flowFacePtr->pNextFlowFaceBlockPtr;
  }


}


/* ____________________________________________________________________ */


int iCheckIfTwoListsAreTheSameF(int listA[3], int listB[3])

{
 int  i,j ;

 int  list1[3], list2[3];

 int  tempInt;


 for(i=0; i<3; i++)
  list1[i] = listA[i];
```

```
                                        /* Sort list1                */
 for(i=0; i<3; i++)
  for(j=i+1; j<3; j++)
   if(list1[i] > list1[j])
    { tempInt = list1[i];
      list1[i] = list1[j];
      list1[j] = tempInt;
    }


 for(i=0; i<3; i++)
  list2[i] = listB[i];

                                        /* Sort list2                */
 for(i=0; i<3; i++)
  for(j=i+1; j<3; j++)
   if(list2[i] > list2[j])
    { tempInt = list2[i];
      list2[i] = list2[j];
      list2[j] = tempInt;
    }

 if( (list1[0] == list2[0]) && (list1[1] == list2[1]) && (list1[2] == list2[2]) )
   return(TRUE);
 else
   return(FALSE);


 }

 /* ____________________________________________________________ */


 PARC_BLOCK checkIfArcExistsF( int list[3] )
```

```
{

 ARC_BLOCK    *arcPtr;

 int  numbersAreTheSame;

 int  i;


 arcPtr = pHeadArcListBlockPtr;

 while(arcPtr != NULL)
  { for(i=0; i<3; i++)
    {
     numbersAreTheSame = iCheckIfTwoListsAreTheSameF(&arcPtr->nodeId[0],list);
     if(numbersAreTheSame)
       return(arcPtr);
    }

   arcPtr = arcPtr->pNextArcBlockPtr;

  }

 return(NULL);
}

/* ____________________________________________________________ */

void
 readElementConnectionFile()
{
 FILE         *inputf;
 char         dummyStr[80];
 COLUMN_BLOCK   *currentColumnPtr;
```

```
NODE_BLOCK    *currentNodePtr;
int           i, lineType;
int           arcCount;
int           nodeInArc[3];
ARC_BLOCK     *arcPtr;
int           numberOfNodesInCell;
int           nodeList[20];
int           index;

                                    /* Open the element to node connection
                                       table file                    */

inputf = fopen("r102.geo","r");

if(inputf == NULL)
 { printf("Unable to open 'r102.geo'\n");
   exit(0);
 }


                                    /* Read till end of file         */
while(1)
  {
    fscanf(inputf,"%s",dummyStr);
                                    /* foef returns a nonzero (TRUE) value
                                       after the FIRST read BEYOND the end
                                       of file!                      */
    if( feof(inputf) ) break;

    lineType = OTHERSTUFF;
                                    /* strcmp returns 0 if args same     */
    if( !strcmp(dummyStr,"GE") )
     lineType = GE;
    else
```

```
{ if( !strcmp(dummyStr,"GNN") )
 lineType = GNN;
}


switch(lineType)

{ case GE:
                                    /* Note that we will number our
                                       elements as they are numbered
                                       in RMA10 for the surface layer.
                          So we read the element ID from
                                       the connection file.          */


     currentColumnPtr = allocateNewColumnBlockF();

     currentColumnPtr->numberOfLayers = NUMBER_OF_LAYERS;

     fscanf(inputf,"%d",&currentColumnPtr->columnId);

     for(i=0; i<8; i++)
      fscanf(inputf,"%d",&nodeList[i]);

     numberOfNodesInCell = 0;
     for(i=0; i<8; i++)
      if(nodeList[i] != 0)
        numberOfNodesInCell++;
                                    /* Repeat first node Id as last node  */
     nodeList[numberOfNodesInCell] = nodeList[0];


     if(numberOfNodesInCell == 8)
      { currentColumnPtr->columnType = TRAPAZOID;
```

```
    currentColumnPtr->arcCount = 4;
  }
 else
  if(numberOfNodesInCell == 6)
   { currentColumnPtr->columnType = TRIANGLE;
    currentColumnPtr->arcCount = 3;
   }

 switch(currentColumnPtr->columnType)
 {
   case TRAPAZOID:

       index = 0;
       for(arcCount=0; arcCount<4; arcCount++)
        {
         for(i=0; i<3; i++)
          {
           if(i != 0) index++;
           nodeInArc[i] = nodeList[index];
          }

         arcPtr = checkIfArcExistsF( nodeInArc );

         if(arcPtr == NULL)
          { arcPtr = allocateNewArcBlockF();
           arcPtr->nodeId[0] = nodeInArc[0];
           arcPtr->nodeId[1] = nodeInArc[1];
           arcPtr->nodeId[2] = nodeInArc[2];
          }

         currentColumnPtr->arcIds[arcCount] = arcPtr->arcId;
        }
       break;
```

```
        case TRIANGLE:

            index = 0;
            for(arcCount=0; arcCount<3; arcCount++)
             {
              for(i=0; i<3; i++)
                {
                 if(i != 0) index++;
                 nodeInArc[i] = nodeList[index];
                }

              arcPtr = checkIfArcExistsF( nodeInArc );

              if(arcPtr == NULL)
               { arcPtr = allocateNewArcBlockF();
                 arcPtr->nodeId[0] = nodeInArc[0];
                 arcPtr->nodeId[1] = nodeInArc[1];
                 arcPtr->nodeId[2] = nodeInArc[2];
               }

              currentColumnPtr->arcIds[arcCount] = arcPtr->arcId;
             }
            break;


     }

    break;
```

```
        case GNN:

                                        /* Note that we will number our
                                           nodes as they are numbered
                                           in RMA10. So we read the node
                                           id  from the connection file.     */

            currentNodePtr = allocateNewNodeBlockF();

            fscanf(inputf,"%d",&currentNodePtr->nodeId);

            for(i=0; i<3; i++)
             fscanf(inputf,"%f",&currentNodePtr->coordinates[i]);

            break;

      }
    }

  fclose(inputf);
}


/* ____________________________________________________________ */


void
 mapColumnsToArcs()

{
```

```
PCOLUMN_BLOCK  columnPtr;
PARC_BLOCK     arcPtr;
int            arcId;
int            i, iTemp;


columnPtr = pHeadColumnListBlockPtr;

while(columnPtr != NULL)
 {
                                        /* Each column has 4 potential arcs   */

 for(i=0; i<columnPtr->arcCount; i++)
  {
   if(columnPtr->arcIds[i] < 1) continue;
   arcId = columnPtr->arcIds[i];
   arcPtr = findArcPtrF(arcId);

                                        /* Set so the smaller column id is the
                                           first in adjacent column list      */


  if(arcPtr->adjacentColumn[1] == -1)
    {
     arcPtr->adjacentColumn[1] = columnPtr->columnId;

    }
   else
    { if(columnPtr->columnId > arcPtr->adjacentColumn[1])
      { arcPtr->adjacentColumn[0] = arcPtr->adjacentColumn[1];
        arcPtr->adjacentColumn[1] = columnPtr->columnId;
      }
     else
```

```
        arcPtr->adjacentColumn[0] = columnPtr->columnId;
      }

    }

   columnPtr = columnPtr->pNextColumnBlockPtr;

  }

}


/* ____________________________________________________________ */

int
 searchColumnListForNamedArc(PCOLUMN_BLOCK columnPtr,
                             int arcId)


{
 PCOLUMN_BLOCK  columnPtr2;

 int  i, foundIt;


 columnPtr2 = pHeadColumnListBlockPtr;

 while( columnPtr2 != NULL)
  {
   if(columnPtr2 == columnPtr)
    { columnPtr2 = columnPtr2->pNextColumnBlockPtr;
      continue;
    }
```

```
    foundIt = FALSE;


    for(i=0; i<columnPtr2->arcCount; i++)
     { if(columnPtr2->arcIds[i] == arcId)
       foundIt = TRUE;
     }


    if( foundIt )
     return(columnPtr2->columnId);

    columnPtr2 = columnPtr2->pNextColumnBlockPtr;
   }

  return(-1);

 }


/* ____________________________________________________________ */

int
 iCheckIfArcIsAFlowBoundaryF(PARC_BLOCK arcPtr)

 {

 PBOUNDARY_NODE_BLOCK  boundaryNodePtr;
 int nodeId, i;
 int foundIt;


 for(i=0; i<3; i++)
  { boundaryNodePtr = pHeadBoundaryNodeBlockPtr;
```

```
   foundIt = NO;

   while( boundaryNodePtr != NULL)
   { nodeId = arcPtr->nodeId[i];
     if(boundaryNodePtr->boundaryNodeId == nodeId)
      { foundIt =YES;
        break;
      }

     boundaryNodePtr = boundaryNodePtr->nextBoundaryNodeBlockPtr;
    }

   if(foundIt == NO)
    return(NO);

 }

 return(YES);
}


/* ____________________________________________________________ */

void determineArcTypeF()
{
 PARC_BLOCK arcPtr;
 int       result;


 arcPtr = pHeadArcListBlockPtr;

 while(arcPtr != NULL)
  { if((arcPtr->adjacentColumn[0] != -1) && (arcPtr->adjacentColumn[1] != -1))
     arcPtr->type = INTERNAL;
```

```
    else
      {  result = iCheckIfArcIsAFlowBoundaryF(arcPtr);
        if(result == YES)
          arcPtr->type = BOUNDARY;
        else
          arcPtr->type = WALL;
      }
    arcPtr = arcPtr->pNextArcBlockPtr;
   }

}

/* ____________________________________________________________ */


void
 mapNeighborColumnsForEachColumn()

{

PCOLUMN_BLOCK   columnPtr;

int         arcCount, arcId;
int         otherColumnId;
int         i, first, second, third;


columnPtr = pHeadColumnListBlockPtr;

while(columnPtr != NULL)

 { for( arcCount=0; arcCount<columnPtr->arcCount; arcCount++)
```

```
    { arcId = columnPtr->arcIds[arcCount];

      otherColumnId = searchColumnListForNamedArc( columnPtr, arcId );

      if(otherColumnId > 0)
        { columnPtr->commonNeighborColumnId[columnPtr->commonNeighborColumnCount] = otherColumnId;
          columnPtr->commonNeighborArcId[columnPtr->commonNeighborColumnCount++] = arcId;
        }
    }

   columnPtr = columnPtr->pNextColumnBlockPtr;

  }

}


/* ____________________________________________________________________ */

int
 iCheckIfCommonNodeInTwoArcsF(PARC_BLOCK arcPtr1, PARC_BLOCK arcPtr2)
{
 int i;

  for(i=0; i<3; i++)
   if( (arcPtr1->nodeId[i] == arcPtr2->nodeId[0])
       ||
      (arcPtr1->nodeId[i] == arcPtr2->nodeId[1])
       ||
      (arcPtr1->nodeId[i] == arcPtr2->nodeId[2]) )
     return(YES);

   return(NO);
}
```

```
/* ____________________________________________________________ */

PARC_BLOCK
findOppositeArcInAColumnF(int columnId, int arcId)

{
PCOLUMN_BLOCK   columnPtr;
PARC_BLOCK      arcPtr, arcPtr2, arcPtr3;
int             nodeId, nodeId2, nodeId3;
PNODE_BLOCK     nodePtr, nodePtr2, nodePtr3;
int             i, result;
int             foundOppositeArc;
int             arcId2;
float           x1, y1, x2, y2;
double          dd;
float           d2, d3;


columnPtr = lookupColumnPtrF(columnId);

switch(columnPtr->columnType)
 {
   case TRAPAZOID:
       arcPtr = findArcPtrF(arcId);

       foundOppositeArc = NO;

       for(i=0; i<4; i++)
        {
         arcId2 = columnPtr->arcIds[i];

         if( arcId != arcId2)
          {
           arcPtr2 = findArcPtrF(arcId2);
```

```
            result = iCheckIfCommonNodeInTwoArcsF(arcPtr,arcPtr2);
            if(result == NO)
             {
              foundOppositeArc = YES;
              break;
             }

           }
         }

        if(foundOppositeArc == YES)
         return(arcPtr2);
        else
         return(NULL);

        break;

      case TRIANGLE:
        arcPtr = findArcPtrF(arcId);

        for(i=0; i<3; i++)
         if(columnPtr->arcIds[i]) break;

        if(i == 0)
         { arcPtr2 = findArcPtrF(columnPtr->arcIds[1]);
           arcPtr3 = findArcPtrF(columnPtr->arcIds[2]);
         }
        else
         if(i == 1)
          { arcPtr2 = findArcPtrF(columnPtr->arcIds[0]);
            arcPtr3 = findArcPtrF(columnPtr->arcIds[2]);
          }
         else
          if(i == 2)
```

```
    { arcPtr2 = findArcPtrF(columnPtr->arcIds[0]);
      arcPtr3 = findArcPtrF(columnPtr->arcIds[1]);
    }

  nodeId = arcPtr->nodeId[1];
  nodeId2 = arcPtr2->nodeId[1];
  nodeId3 = arcPtr3->nodeId[1];

  nodePtr = findNodePtrF(nodeId);
  x1 = nodePtr->coordinates[0];
  y1 = nodePtr->coordinates[1];

  nodePtr2 = findNodePtrF(nodeId2);
  x2 = nodePtr->coordinates[0];
  y2 = nodePtr->coordinates[1];

  dd = (x1-x2)*(x1-x2) + (y1-y2)*(y1-y2);
  dd = sqrt(dd);
  d2 = dd;

  nodePtr3 = findNodePtrF(nodeId3);
  x2 = nodePtr->coordinates[0];
  y2 = nodePtr->coordinates[1];

  dd = (x1-x2)*(x1-x2) + (y1-y2)*(y1-y2);
  dd = sqrt(dd);
  d3 = dd;

  if(d2 < d3)
    return(arcPtr2);
  else
    return(arcPtr3);
```

```
  }
                                        /* Catch-All                */
 return(NULL);
  |
}

/* ____________________________________________________________________ */

void
 mapUpstreamAndDownstreamColumnsForEachColumnF()
{

 PCOLUMN_BLOCK   columnPtr;
 int             i, otherColumnId, otherArcId;
 PARC_BLOCK      farArcPtr;


 columnPtr = pHeadColumnListBlockPtr;

 while(columnPtr != NULL)
  {
   for(i=0; i<columnPtr->arcCount; i++)
    { otherColumnId = columnPtr->commonNeighborColumnId[i];
      otherArcId = columnPtr->commonNeighborArcId[i];
      if(otherColumnId != -1)
       {  farArcPtr = findOppositeArcInAColumnF(otherColumnId, otherArcId);
          if(farArcPtr->adjacentColumn[0] == otherColumnId)
            columnPtr->commonNeighbor2ColumnId[i] = farArcPtr->adjacentColumn[1];
          else
          columnPtr->commonNeighbor2ColumnId[i] = farArcPtr->adjacentColumn[0];
       }
      else
          columnPtr->commonNeighbor2ColumnId[i] = -1;
    }
```

```
    columnPtr = columnPtr->pNextColumnBlockPtr;
  }
  !
}
/* ____________________________________________________________ */


void
 createElementList()
{

 ELEMENT_BLOCK *elementPtr;
 ELEMENT_BLOCK *elementAbovePtr;
 COLUMN_BLOCK  *columnPtr;

 int  iMaximumNumberOfLayers;
 int  iCurrentCellLayerId;
 int  iNumberOfCells;
 int  i;
 int  lastRma10ElementId;

                                  /* Determine maximum number of layers
                                     in column list                */
iMaximumNumberOfLayers = 0;

columnPtr = pHeadColumnListBlockPtr;

while(columnPtr != NULL)
 {
  if(columnPtr->numberOfLayers > iMaximumNumberOfLayers)
   iMaximumNumberOfLayers = columnPtr->numberOfLayers;

  columnPtr = columnPtr->pNextColumnBlockPtr;
 }
```

```
                                        /* Do surface layer first          */
iCurrentCellLayerId = 1;
iNumberOfCells = 0;
                                        /* Column IDs same as element IDs in
                                           surface layer.                  */

columnPtr = pHeadColumnListBlockPtr;

while(columnPtr != NULL)
 {
   elementPtr = allocateNewElementBlockF();

   columnPtr->elementPtr = elementPtr;

   columnPtr->surfaceElementPtr = elementPtr;

   elementPtr->columnId = columnPtr->columnId;

   elementPtr->elementType = columnPtr->columnType;

   elementPtr->arcCount = columnPtr->arcCount;

   elementPtr->layerId = iCurrentCellLayerId;

   elementPtr->elementId = columnPtr->columnId;

   elementPtr->rma10ElementId = elementPtr->elementId;

   iNumberOfCells++;

   columnPtr = columnPtr->pNextColumnBlockPtr;
 }

lastRma10ElementId = elementPtr->rma10ElementId;
```

```
                                    /* Now do all subsurface layers      */
for(iCurrentCellLayerId = 2;
   iCurrentCellLayerId<=iMaximumNumberOfLayers;
  iCurrentCellLayerId++)
{

  columnPtr = pHeadColumnListBlockPtr;

  while(columnPtr != NULL)
   {
    if(iCurrentCellLayerId <= columnPtr->numberOfLayers )
     {
      elementPtr = allocateNewElementBlockF();

      elementPtr->columnId = columnPtr->columnId;

      elementPtr->elementType = columnPtr->columnType;

      elementPtr->arcCount = columnPtr->arcCount;

      elementAbovePtr = columnPtr->elementPtr;

      elementPtr->elementAbovePtr = elementAbovePtr;

      elementAbovePtr->elementBelowPtr = elementPtr;

      columnPtr->elementPtr = elementPtr;

      elementPtr->layerId = iCurrentCellLayerId;

      iNumberOfCells++;

      elementPtr->elementId = iNumberOfCells;
```

```
      }

      columnPtr = columnPtr->pNextColumnBlockPtr;

    }

  }

                                    /* Now number all elements with their
                                       rma10ElementId                    */

columnPtr = pHeadColumnListBlockPtr;

while(columnPtr != NULL)
 {
  elementPtr = columnPtr->surfaceElementPtr;
  while(elementPtr->elementBelowPtr != NULL)
   elementPtr = elementPtr->elementBelowPtr;

  while(elementPtr->elementAbovePtr != NULL)
   { elementPtr->rma10ElementId  = ++lastRma10ElementId;
     elementPtr = elementPtr->elementAbovePtr;
   }

  columnPtr = columnPtr->pNextColumnBlockPtr;
 }

}

/* ____________________________________________________________ */

void
 mapElementToNeighborElements()
```

```
{

PCOLUMN_BLOCK   columnPtr, otherColumnPtr, farColumnPtr;
int         columnId, otherColumnId, farColumnId;

int         i;

int         currentLayerId;
int         iMaximumNumberOfLayers;

PELEMENT_BLOCK  elementPtr, otherElementPtr, farElementPtr;


                                    /* Determine maximum number of layers
                                       in column list                */
iMaximumNumberOfLayers = 0;

columnPtr = pHeadColumnListBlockPtr;

while(columnPtr != NULL)
 {
  if(columnPtr->numberOfLayers > iMaximumNumberOfLayers)
   iMaximumNumberOfLayers = columnPtr->numberOfLayers;

  columnPtr = columnPtr->pNextColumnBlockPtr;
 }



currentLayerId = 1;

while(currentLayerId <= iMaximumNumberOfLayers)
 {
   columnPtr = pHeadColumnListBlockPtr;
```

```
while(columnPtr != NULL)
 {
 elementPtr = columnPtr->surfaceElementPtr;

 while(elementPtr->layerId < currentLayerId)
  elementPtr = elementPtr->elementBelowPtr;

 if(elementPtr != NULL)
  {

   for(i=0; i<columnPtr->arcCount; i++)
    {
     otherColumnId = columnPtr->commonNeighborColumnId[i];
     if(otherColumnId != -1)
      {
       otherColumnPtr = lookupColumnPtrF(otherColumnId);
       otherElementPtr = otherColumnPtr->surfaceElementPtr;

       while(otherElementPtr->layerId < currentLayerId)
        otherElementPtr = otherElementPtr->elementBelowPtr;

       if(otherElementPtr != NULL)
        elementPtr->commonNeighborElementId[i] = otherElementPtr->elementId;
       else
        elementPtr->commonNeighborElementId[i] = -1;

       farColumnId = columnPtr->commonNeighbor2ColumnId[i];
       if(farColumnId != -1)
        {
         farColumnPtr = lookupColumnPtrF(farColumnId);
         farElementPtr = farColumnPtr->surfaceElementPtr;

         while(farElementPtr->layerId < currentLayerId)
          farElementPtr = farElementPtr->elementBelowPtr;
```

```
                if(farElementPtr != NULL)
                  elementPtr->commonNeighbor2ElementId[i] = farElementPtr->elementId;
                else
                  elementPtr->commonNeighbor2ElementId[i] = -1;
              }

            }

          }

        }

        columnPtr = columnPtr->pNextColumnBlockPtr;
      }

    currentLayerId++;
  }

}

/* ____________________________________________________________ */

void
generateHorizontalFlowFaceListF()

{
PCOLUMN_BLOCK   columnPtr;
int             iMaximumNumberOfLayers;
int             currentLayerId;
PARC_BLOCK      arcPtr;
PFLOWFACE_BLOCK flowFacePtr;
PELEMENT_BLOCK  elementPtr;
```

```
int         result;
int         columnId;


                                    /* Determine maximum number of layers
                                       in column list              */
iMaximumNumberOfLayers = 0;

columnPtr = pHeadColumnListBlockPtr;

while(columnPtr != NULL)
 {
  if(columnPtr->numberOfLayers > iMaximumNumberOfLayers)
   iMaximumNumberOfLayers = columnPtr->numberOfLayers;

  columnPtr = columnPtr->pNextColumnBlockPtr;
 }



currentLayerId = 1;

while(currentLayerId <= iMaximumNumberOfLayers)
 {
  arcPtr = pHeadArcListBlockPtr;

  while(arcPtr != NULL)
   {

    flowFacePtr = allocateNewFlowFaceBlockF();

    flowFacePtr->orientation = HORIZONTAL;

    flowFacePtr->layerId = currentLayerId;
```

```
flowFacePtr->arcPtr = arcPtr;

flowFacePtr->adjacentColumn[0] = arcPtr->adjacentColumn[0];
flowFacePtr->adjacentColumn[1] = arcPtr->adjacentColumn[1];

columnId = arcPtr->adjacentColumn[0];
if(columnId != -1)
  { columnPtr = lookupColumnPtrF(columnId);
    elementPtr = columnPtr->surfaceElementPtr;

    while(elementPtr->layerId <currentLayerId)
      elementPtr = elementPtr->elementBelowPtr;

    if(elementPtr != NULL)
      flowFacePtr->adjacentElement[0] = elementPtr->elementId;
    else
      flowFacePtr->adjacentElement[0] = -1;
    }

columnId = arcPtr->adjacentColumn[1];
if(columnId != -1)
  { columnPtr = lookupColumnPtrF(columnId);
    elementPtr = columnPtr->surfaceElementPtr;

    while(elementPtr->layerId <currentLayerId)
      elementPtr = elementPtr->elementBelowPtr;

    if(elementPtr != NULL)
      flowFacePtr->adjacentElement[1] = elementPtr->elementId;
    else
      flowFacePtr->adjacentElement[1] = -1;
    }

switch(arcPtr->type)
```

```
        {
          case INTERNAL:
              flowFacePtr->type = INTERNAL;
              break;

          case BOUNDARY:
              flowFacePtr->type = BOUNDARY;
              break;

          case WALL:
              flowFacePtr->type = WALL;
              break;
        }

       arcPtr = arcPtr->pNextArcBlockPtr;
     }

   currentLayerId++;

  }


}

/* ____________________________________________________________ */

void
 mapHorizontalFlowFaces()

{
 PELEMENT_BLOCK   elementPtr;
 PFLOWFACE_BLOCK  flowFacePtr;
 PARC_BLOCK       arcPtr, oppositeArcPtr;
```

```
int         foundIt, i;
int         targetElementId;

int         leftElementId,   rightElementId;
PELEMENT_BLOCK  leftElementPtr,  rightElementPtr;
int         leftColumnId,    rightColumnId;
int         farLeftColumnId, farRightColumnId;
PCOLUMN_BLOCK   farLeftColumnPtr, farRightColumnPtr;
PELEMENT_BLOCK  farElementPtr;
int         farLeftElementId, farRightElementId;


flowFacePtr = pHeadFlowFaceListBlockPtr;

while(flowFacePtr != NULL)
 {
  switch(flowFacePtr->type)
   {
     case INTERNAL:
        leftElementId = flowFacePtr->adjacentElement[0];
        rightElementId = flowFacePtr->adjacentElement[1];

        flowFacePtr->upstreamElementId =  leftElementId;
        flowFacePtr->downstreamElementId =  rightElementId;

        leftElementPtr = lookupElementPtrF(leftElementId);

        foundIt = NO;
        for(i=0; i<leftElementPtr->arcCount; i++)
         {
          targetElementId = leftElementPtr->commonNeighborElementId[i];
          if(targetElementId == rightElementId)
           { foundIt = YES;
             break;
```

```
    }
  }

 if(foundIt == YES)
  {
   flowFacePtr->farDownstreamElementId =
   leftElementPtr->commonNeighbor2ElementId[i];
  }
 else
  flowFacePtr->farDownstreamElementId = -1;


  rightElementPtr = lookupElementPtrF(rightElementId);

  foundIt = NO;
  for(i=0; i<rightElementPtr->arcCount; i++)
   {
    targetElementId = rightElementPtr->commonNeighborElementId[i];
    if(targetElementId == leftElementId)
     { foundIt = YES;
       break;
     }
   }

  if(foundIt == YES)
   {
    flowFacePtr->farUpstreamElementId =
    rightElementPtr->commonNeighbor2ElementId[i];
   }
  else
   flowFacePtr->farUpstreamElementId = -1;

  break;
```

```
case BOUNDARY:
    leftElementId = flowFacePtr->adjacentElement[0];
    rightElementId = flowFacePtr->adjacentElement[1];

    flowFacePtr->upstreamElementId =  leftElementId;
    flowFacePtr->downstreamElementId =  rightElementId;

    if(leftElementId == -1)
     flowFacePtr->farUpstreamElementId = -1;
    else
     { arcPtr = flowFacePtr->arcPtr;

      leftElementPtr = lookupElementPtrF(leftElementId);
      leftColumnId = leftElementPtr->columnId;

      oppositeArcPtr =  findOppositeArcInAColumnF(leftColumnId, arcPtr->arcId);
      if(oppositeArcPtr->adjacentColumn[0] == leftColumnId)
       farLeftColumnId = oppositeArcPtr->adjacentColumn[1];
      else
       farLeftColumnId = oppositeArcPtr->adjacentColumn[0];

      farLeftColumnPtr = lookupColumnPtrF(farLeftColumnId);
      farElementPtr = farLeftColumnPtr->surfaceElementPtr;

      while(farElementPtr->layerId < leftElementPtr->layerId)
       farElementPtr = farElementPtr->elementBelowPtr;

      if(farElementPtr != NULL)
       farLeftElementId = farElementPtr->elementId;
      else
       farLeftElementId = -1;

      flowFacePtr->farUpstreamElementId = farLeftElementId;
```

```
 }

if(rightElementId == -1)
  flowFacePtr->farDownstreamElementId = -1;
else
 { arcPtr = flowFacePtr->arcPtr;

  rightElementPtr = lookupElementPtrF(rightElementId);
  rightColumnId = rightElementPtr->columnId;

  oppositeArcPtr =  findOppositeArcInAColumnF(rightColumnId,
              arcPtr->arcId);

  if(oppositeArcPtr->adjacentColumn[0] == rightColumnId)
    farRightColumnId = oppositeArcPtr->adjacentColumn[1];
  else
    farRightColumnId = oppositeArcPtr->adjacentColumn[0];

  farRightColumnPtr = lookupColumnPtrF(farRightColumnId);
  farElementPtr = farRightColumnPtr->surfaceElementPtr;

  while(farElementPtr->layerId < rightElementPtr->layerId)
    farElementPtr = farElementPtr->elementBelowPtr;

  if(farElementPtr != NULL)
    farRightElementId = farElementPtr->elementId;
  else
    farRightElementId = -1;

  flowFacePtr->farDownstreamElementId = farRightElementId;

 }

break;
```

```
        case WALL:
            break;

    }


    flowFacePtr = flowFacePtr->pNextFlowFaceBlockPtr;
   }

}

/* ____________________________________________________________ */

void
 mapVerticalFlowFaces()

{
 PELEMENT_BLOCK   elementPtr;
 PFLOWFACE_BLOCK  flowFacePtr;

 PCOLUMN_BLOCK    columnPtr;
 PELEMENT_BLOCK   farUpstreamElementPtr,
                  upstreamElementPtr,
                  downstreamElementPtr,
                  farDownstreamElementPtr;


                                  /* For FEM RMA10 .....
                                    POSITIVE FLOW IS TAKEN TO BE
                                    UP! IRREGARDLESS of other
                                    conventions                  */
```

```
                                        /* ICM will have vertical flow faces
                                           numbered from the bottom up with
                                           and POSITIVE flow taken to be up   */
columnPtr = pHeadColumnListBlockPtr;

while(columnPtr != NULL)
 {

  elementPtr = columnPtr->surfaceElementPtr;
  while(elementPtr->elementBelowPtr != NULL)
   elementPtr = elementPtr->elementBelowPtr;

  downstreamElementPtr = elementPtr->elementAbovePtr;

  while(downstreamElementPtr != NULL)
   {
    farDownstreamElementPtr = downstreamElementPtr->elementAbovePtr;
    upstreamElementPtr = downstreamElementPtr->elementBelowPtr;
    farUpstreamElementPtr = upstreamElementPtr->elementBelowPtr;

    flowFacePtr = allocateNewFlowFaceBlockF();
    flowFacePtr->orientation = VERTICAL;

    flowFacePtr->type= INTERNAL;
    downstreamElementPtr->bottomFlowFaceId = flowFacePtr->flowFaceId;


    if(farUpstreamElementPtr != NULL)
     flowFacePtr->farUpstreamElementId = farUpstreamElementPtr->elementId;
    else
     flowFacePtr->farUpstreamElementId = -1;

    flowFacePtr->upstreamElementId = upstreamElementPtr->elementId;
    flowFacePtr->downstreamElementId  = downstreamElementPtr->elementId;
```

```
      if(farDownstreamElementPtr != NULL)
        flowFacePtr->farDownstreamElementId = farDownstreamElementPtr->elementId;
       else
        flowFacePtr->farDownstreamElementId = -1;

       downstreamElementPtr = downstreamElementPtr->elementAbovePtr;
      }

    columnPtr = columnPtr->pNextColumnBlockPtr;
   }


}



/* ____________________________________________________________ */


PARC_BLOCK
 findArcPtrF(int arcId)

{

 PARC_BLOCK arcPtr;

 arcPtr = pHeadArcListBlockPtr;

 while(arcPtr != NULL)
  {

   if(arcPtr->arcId == arcId)
    return(arcPtr);
```

```
    arcPtr = arcPtr->pNextArcBlockPtr;
   }

  return(NULL);

}

/* ____________________________________________________________ */


PNODE_BLOCK
 findNodePtrF(int nodeId)

{

 PNODE_BLOCK nodePtr;

 nodePtr = pHeadNodeBlockPtr;

 while(nodePtr != NULL)
  {

   if(nodePtr->nodeId == nodeId)
    return(nodePtr);

   nodePtr = nodePtr->nextNodeBlockPtr;
  }

 return(NULL);

}

/* ____________________________________________________________ */
```

```
                                /* DO NOT USE-- node numbering SKIPS around   */
PNODE_BLOCK
 lookupNodePtrF(int nodeId)
   !
{

 PNODE_BLOCK nodePtr;


 if(lookupNodePtrArray == NULL)
  {

    lookupNodePtrArray =
     (PNODE_BLOCK *)calloc(globalNodeCount+1,sizeof(PNODE_BLOCK));

    nodePtr = pHeadNodeBlockPtr;

    while(nodePtr != NULL)
     {
      lookupNodePtrArray[nodePtr->nodeId] = nodePtr;

      nodePtr = nodePtr->nextNodeBlockPtr;
     }

  }

 if(nodeId > globalNodeCount)
  return(NULL);
 else
  return(lookupNodePtrArray[nodeId]);


}
```

```
/* ____________________________________________________________ */

PCOLUMN_BLOCK
 findColumnPtr(int columnId)

{

 PCOLUMN_BLOCK columnPtr;


 columnPtr = pHeadColumnListBlockPtr;

 while(columnPtr != NULL)
  {

   if(columnPtr->columnId == columnId)
    return(columnPtr);

   columnPtr = columnPtr->pNextColumnBlockPtr;
  }

 return(NULL);

}

/* ____________________________________________________________ */


PCOLUMN_BLOCK
 lookupColumnPtrF(int columnId)

{
```

```
PCOLUMN_BLOCK columnPtr;

if (lookupColumnPtrArray == NULL)
{

   lookupColumnPtrArray =
     (PCOLUMN_BLOCK *)calloc (globalColumnCount+1,sizeof (PCOLUMN_BLOCK ) ) ;

   ColumnPtr = pHeadColumnListBlockPtr;

    while ( columnPtr != NULL )
      {
        lookupColumnPtrArray[columnPtr -> columnId] = columnPtr;

        ColumnPtr = columnPtr ->pNextColumnBlockPtr ;
       }

}

if ( (columnId < 1)  || (columnId > globalColumncount ) )
   return (NULL) ;
else
   return ( lookupColumnPtrArray[columnId] ) ;

}

/* ____________________________________________________________ */

PELEMENT_BLOCK
  LookupElementPtrF ( int elementId)

 {

 PELEMENT_BLOCK elementPtr;
```

```
if(lookupElementPtrArray == NULL)
 {

  lookupElementPtrArray =
   (PELEMENT_BLOCK *)calloc(globalElementCount+1,sizeof(PELEMENT_BLOCK));

  elementPtr = pHeadElementListBlockPtr;

  while(elementPtr != NULL)
   {
    lookupElementPtrArray[elementPtr->elementId] = elementPtr;

    elementPtr = elementPtr->pNextElementBlockPtr;
   }

 }

if( (elementId < 1) || (elementId > globalElementCount))
 return(NULL);
else
 return(lookupElementPtrArray[elementId]);

}


/* ____________________________________________________________ */


PFLOWFACE_BLOCK

lookupFlowFacePtrF(int flowFaceId)
```

```
{

PFLOWFACE_BLOCK flowFacePtr;


if(lookupFlowFacePtrArray == NULL)
  {

   lookupFlowFacePtrArray =
    (PFLOWFACE_BLOCK *)calloc(globalFlowFaceCount+1,sizeof(PFLOWFACE_BLOCK));

   flowFacePtr = pHeadFlowFaceListBlockPtr;

   while(flowFacePtr != NULL)
    {
     lookupFlowFacePtrArray[flowFacePtr->flowFaceId] = flowFacePtr;

     flowFacePtr = flowFacePtr->pNextFlowFaceBlockPtr;
    }

  }

if( (flowFaceId < 1) || (flowFaceId > globalFlowFaceCount))
  return(NULL);
else
  return(lookupFlowFacePtrArray[flowFaceId]);

}



/* ____________________________________________________________ */
```

```
void
 setFlowFaceIdentifierTags()
{

 PFLOWFACE_BLOCK  flowFacePtr;
 PARC_BLOCK      arcPtr;
 int             surfaceCenterNodeId;
 int             topCenterNodeId;
 int             layerFlowFaceIsIn;
 PELEMENT_BLOCK   elementPtr;
 int             elementId;

 flowFacePtr = pHeadFlowFaceListBlockPtr;

 while(flowFacePtr != NULL)
 {
  if(flowFacePtr->orientation == HORIZONTAL)
  {
   arcPtr = flowFacePtr->arcPtr;

   surfaceCenterNodeId = arcPtr->nodeId[1];

   layerFlowFaceIsIn = flowFacePtr->layerId;

   if(layerFlowFaceIsIn == 1)
    topCenterNodeId = surfaceCenterNodeId;
   else
    topCenterNodeId =
     surfaceNodeList[surfaceCenterNodeId].nref + (layerFlowFaceIsIn-1);

   flowFacePtr->identifierTag = topCenterNodeId;
  }
  else
   if(flowFacePtr->orientation == VERTICAL)
```

```
    {
      elementId =flowFacePtr->upstreamElementId;
      elementPtr = lookupElementPtrF(elementId);
      flowFacePtr->identifierTag = elementPtr->rma10ElementId;

                                        /* Check if upstream cell is a surface
                                          cell. If it is then the identifier
                                          is taken to be the flow face's
                                          downstream element's rma10ElementId.
                                          IF the flow faces upstream element
                                          is not in the surface layer, then
                                          it must be in a layer beneath the
                                          surface layer. All vertical flow
                                          faces beneath the first vertical
                                          flow face in a column WILL HAVE THE
                                          WRONG SENSE OF FLUX when the fluxes
                                          are read in from the hydrodynamic
                                          file. This is because in RMA10 they
                                          number elements first in the surface,
                                          then they start numbering elements
                                          in the subsurface at the bottom
                                          layer then go up in the same column
                                          to the 2nd layer! The flux sense
                                          in BOTH models is taken to be from
                                          an element with the smaller element
                                          id to the larger element id. We will
                                          indicate a change in flux sign by
                                          making the identifier tag NEGATIVE */

    }

  flowFacePtr = flowFacePtr->pNextFlowFaceBlockPtr;
}
```

```
}


/* ____________________________________________________________ */

void
renumberAllFlowFacesForOutputF()

{

 PFLOWFACE_BLOCK  flowFacePtr;
 int              modelFlowFaceId;
 char             faceType, faceOrientation;


 flowFacePtr = pHeadFlowFaceListBlockPtr;

 modelFlowFaceId = 1;

 while(flowFacePtr != NULL)
  {
   switch(flowFacePtr->type)
    {  case INTERNAL:
          faceType = 'I';
          break;

       case BOUNDARY:
          faceType = 'B';
          break;

       case WALL:
          faceType = 'W';
          break;
    }
```

```
    switch(flowFacePtr->orientation)
     { case  HORIZONTAL:
           faceOrientation = 'H';
           break;

       case  VERTICAL:
           faceOrientation = 'V';
           break;
     }


    if( (faceType == 'B') || (faceType == 'I'))
     {
       flowFacePtr->modelFlowFaceId = modelFlowFaceId++;

     }


    flowFacePtr = flowFacePtr->pNextFlowFaceBlockPtr;
   }



}

/* ____________________________________________________________ */

int
 *retriveAColumnsVerticalFlowFaceInfo( PCOLUMN_BLOCK    columnPtr )

{
 PELEMENT_BLOCK   elementPtr;
 PFLOWFACE_BLOCK  flowFacePtr;
```

```
PELEMENT_BLOCK   upstreamElementPtr;

int          i, flowFaceId;
int          columnFlowFaceCount = 0;

int          *intList;
int          intListIndex = 0;


 elementPtr = columnPtr->surfaceElementPtr;
 while(elementPtr->elementBelowPtr != NULL)
  { columnFlowFaceCount++;
    elementPtr = elementPtr->elementBelowPtr;
  }
                                          /* Make first element in array
                                             be count of total size of
                                             array.              */

 intList = (int *)calloc(columnFlowFaceCount+1,sizeof(int));
 intList[intListIndex++] = columnFlowFaceCount;

 upstreamElementPtr = elementPtr->elementAbovePtr;

 while(upstreamElementPtr != NULL)
  {
   flowFaceId = upstreamElementPtr->bottomFlowFaceId;
   intList[intListIndex++] = flowFaceId;
   upstreamElementPtr = upstreamElementPtr->elementAbovePtr;
  }


return(intList);

}
```

```
/* ____________________________________________________________ */

void
 printFlowFaceListF()

 {

 PELEMENT_BLOCK   elementPtr;
 FLOWFACE_BLOCK   *flowFacePtr;
 int              flowFaceId;
 FILE               *outf;
 char              faceType, faceOrientation;
 int              modelFlowFaceId;
 int              upstreamElementId,
                  downstreamElementId,
                  farUpstreamElementId,
                  farDownstreamElementId;

 PCOLUMN_BLOCK    columnPtr;
 int              *intList;
 int              i, numberInList;



 outf = fopen("FF.out","w");


 fprintf(outf,"    ICM    FEM\n");
 elementPtr = pHeadElementListBlockPtr;

 while(elementPtr != NULL)
  {
   fprintf(outf,"%8d  %6d\n",
        elementPtr->elementId,
```

```
        elementPtr->rma10ElementId);

  elementPtr = elementPtr->pNextElementBlockPtr;
 }

fprintf(outf,"\n   FFID    ILB     IL     JR    JRB  TYPE ORIENT    TAG\n");

flowFacePtr = pHeadFlowFaceListBlockPtr;

flowFaceId = 0;
while(flowFacePtr != NULL)
 {
  switch(flowFacePtr->type)
   {  case INTERNAL:
         faceType = 'I';
         break;

      case BOUNDARY:
         faceType = 'B';
         break;

      case WALL:
         faceType = 'W';
         break;
   }

  switch(flowFacePtr->orientation)
   { case  HORIZONTAL:
         faceOrientation = 'H';
         break;

     case  VERTICAL:
         faceOrientation = 'V';
         break;
```

```
  }

modelFlowFaceId = flowFacePtr->modelFlowFaceId;

if(flowFacePtr->farUpstreamElementId != -1)
 farUpstreamElementId = flowFacePtr->farUpstreamElementId;
else
 farUpstreamElementId = 0;

if(flowFacePtr->upstreamElementId != -1)
 upstreamElementId = flowFacePtr->upstreamElementId;
else
 upstreamElementId = 0;

if(flowFacePtr->downstreamElementId != -1)
 downstreamElementId = flowFacePtr->downstreamElementId;
else
 downstreamElementId = 0;

if(flowFacePtr->farDownstreamElementId != -1)
 farDownstreamElementId = flowFacePtr->farDownstreamElementId;
else
 farDownstreamElementId = 0;


if( (flowFacePtr->type == BOUNDARY) || (flowFacePtr->type == INTERNAL))
 {
   flowFaceId++;
   fprintf(outf,"%8d %6d %6d %6d %6d    %c     %c %6d\n",
         modelFlowFaceId,
         farUpstreamElementId,
         upstreamElementId,
         downstreamElementId,
         farDownstreamElementId,
```

```
            faceType,
            faceOrientation,
            flowFacePtr->identifierTag);
    }

   flowFacePtr = flowFacePtr->pNextFlowFaceBlockPtr;
  }


                                        /* Write out the number of vertical
                                           flow faces in a column info       */

fprintf(outf,"\n     SB    NVF\n");

columnPtr = pHeadColumnListBlockPtr;

while(columnPtr != NULL)
 {
                                        /* Function returns a list of vertical
                                           flow faces for each column. This
                                           list is dynamically allocated so
                                           need to release memory when we
                              finish with it.                */

   intList = retriveAColumnsVerticalFlowFaceInfo(columnPtr);
   numberInList = intList[0];

   fprintf(outf,"%8d%8d\n",columnPtr->columnId, numberInList);

   free(intList);

   columnPtr = columnPtr->pNextColumnBlockPtr;
 }
```

```
                                        /* Write out the  vertical
                                           flow faces in a column info      */

fprintf(outf,"\n     SB     VFN LIST...\n");

columnPtr = pHeadColumnListBlockPtr;

while(columnPtr != NULL)
 {
                                        /* Function returns a list of vertical
                                           flow faces for each column. This
                                           list is dynamically allocated so
                                           need to release memory when we
                              finish with it.                */

  intList = retriveAColumnsVerticalFlowFaceInfo(columnPtr);
  numberInList = intList[0];

  fprintf(outf,"  %6d",columnPtr->columnId);
  for(i=1; i<=numberInList; i++)
   { flowFacePtr = lookupFlowFacePtrF(intList[i]);
     modelFlowFaceId = flowFacePtr->modelFlowFaceId;
     fprintf(outf,"  %6d",modelFlowFaceId);
   }
  fprintf(outf,"\n");

  free(intList);

  columnPtr = columnPtr->pNextColumnBlockPtr;
 }



fclose(outf);
```

```
}


/* ____________________________________________________________ */

void
 printGeoFileF()

{

 FILE          *outf;
 PCOLUMN_BLOCK   columnPtr;
 PELEMENT_BLOCK  elementPtr;
 PELEMENT_BLOCK  surfaceElementPtr, bottomElementPtr;
 PELEMENT_BLOCK  elementAbovePtr;
 int           elementAboveId;
 PFLOWFACE_BLOCK  flowFacePtr;
 char          faceType;

 float         BIL, BI, BID, BIR;


 outf = fopen("geo.out","w");

                                    /* Write out for each column, the
                                       surface element id and its
                                       corresponding bottom element id   */

 fprintf(outf,"   SBN    BBN\n");

 columnPtr = pHeadColumnListBlockPtr;

 while(columnPtr != NULL)
```

```
  {

   elementPtr = columnPtr->surfaceElementPtr;
   surfaceElementPtr = elementPtr;

   while(elementPtr->elementBelowPtr != NULL)
    elementPtr = elementPtr->elementBelowPtr;

   bottomElementPtr = elementPtr;

   fprintf(outf,
         "%6d %6d\n",
         surfaceElementPtr->elementId,
         bottomElementPtr->elementId);

   columnPtr = columnPtr->pNextColumnBlockPtr;
  }


                                    /* Now print out boxes above        */
fprintf(outf,"\n    BU\n");

elementPtr = pHeadElementListBlockPtr;

while(elementPtr != NULL)
 {
  elementAbovePtr = elementPtr->elementAbovePtr;
  if(elementAbovePtr != NULL)
    elementAboveId = elementAbovePtr->elementId;
  else
    elementAboveId = 0;

  fprintf(outf,"%6d \n",elementAboveId);
```

```
  elementPtr = elementPtr->pNextElementBlockPtr;
 }



                                    /* Now print out centroid ofsets     */

fprintf(outf,"\n      BIL       BI       BID       BIR       type    layer\n");

flowFacePtr =  pHeadFlowFaceListBlockPtr;
while( (flowFacePtr != NULL) && (flowFacePtr->orientation == HORIZONTAL) )
 {
  if( (flowFacePtr->type == BOUNDARY) || (flowFacePtr->type == INTERNAL))
   {

    switch(flowFacePtr->type)
     {  case INTERNAL:
           faceType = 'I';
           break;

       case BOUNDARY:
           faceType = 'B';
           break;

       case WALL:
           faceType = 'W';
           break;
     }

    BIL = flowFacePtr->distanceBetweenUpstreamCentroids;
    BI  = flowFacePtr->distanceBetweenCentroids;
    BID = flowFacePtr->downstreamDistanceBetweenCentroidAndFace;
    BIR = flowFacePtr->distanceBetweenDownstreamCentroids;
```

```
      fprintf(outf,
            " %10.5e %10.5e %10.5e %10.5e      %c     %3d\n",
            BIL,BI,BID,BIR,faceType,flowFacePtr->layerId);

  /*  fprintf(outf," %15.9e %15.9e %15.9e %15.9e\n",BIL,BI,BID,BIR); */

     }

    flowFacePtr = flowFacePtr->pNextFlowFaceBlockPtr;

   }


  fclose(outf);

}


/* ____________________________________________________________ */

void  readBoundaryArcFile()
{


 FILE     *infile;
 int      i, int1, int2, index;
 char     lineBuffer[132];
 int      NP, NE, NPM, NEM;
 int      surfaceNodeId;
 int      ndep, nref;

 BOUNDARY_NODE_BLOCK  *boundaryNodePtr;
```

```
infile = fopen("r4icr10.output","r");
if(infile == NULL)
 { printf("Unable to open 'r4icr10.output'\n");
   fflush(stdout);
   exit(0);
 }


                                        /* Skip first 2 lines          */
fgets(lineBuffer,132,infile);
fgets(lineBuffer,132,infile);

fscanf(infile,"%d %d %d %d", &NP, &NE, &NPM, &NEM);

if(NPM > 100000)
 { printf("number of nodes in surface layer is 100000!\n");
   printf("If thats true then take out this line and recompile!\n");
   fflush(stdout);
   exit(0);
 }

surfaceNodeList =
   (SURFACE_NODE_LIST *)calloc(NPM+1,sizeof(SURFACE_NODE_LIST));

                                        /* Skip first next 4 lines     */
for(i=0; i<4; i++)
 fgets(lineBuffer,132,infile);

for(i=0; i<NPM; i++)
 { fscanf(infile,"%d %d %d", &surfaceNodeId, &ndep, &nref);
   surfaceNodeList[surfaceNodeId].ndep = ndep;
   surfaceNodeList[surfaceNodeId].nref = nref;
 }
```

```
                                    /* Skip first next 3 lines          */
 for(i=0; i<3; i++)
  fgets(lineBuffer,132,infile);

 fscanf(infile,"%d %d",&int1, &int2);
 while(!feof(infile))
  {
   if(int1 < 374)
    if( (int2 == 31000) || (int2 == 200) || (int2 == 1200) )
     {
      boundaryNodePtr = allocateNewBoundaryNodeBlockF();
      boundaryNodePtr->boundaryNodeId = int1;
     }

    fscanf(infile,"%d %d",&int1, &int2);

   }


}

/* ____________________________________________________________________ */
```

Appendix B
FEMCONVT Source Code Listing

```
C     THIS PROGRAM CALCULATES THE FLUXES THROUGH THE
C     FACES OF A RMA10 BINARY
C     OUTPUT FILE.  IT ALSO CALCULATES THEIR VOLUME.
      Parameter (MNP=2000,MEL=2000,NLAYM=4)
      CHARACTER CHFN*15
      DIMENSION IDND(MNP,8),NNPE(MEL),LIS(MNP),FLUX(MNP)
     .      ,AREA(MNP)
      DIMENSION LISE(MEL),VOL(MEL),VAREA(MEL),VFLUX(MEL)
      DIMENSION CORD(MNP,3),SPEC(MNP,3),ALFA(MNP),NFIX(MNP)
     . ,AO(MNP),NSURF(MNP),NDEP(MNP),NREF(MNP),NOP(MEL,20)
     . ,NCORN(MEL),IMAT(MEL),TH(MEL),NFIXH(MEL),WIDTH(MNP)
      DIMENSION XVEL(4,MNP),VVEL(MNP),DFCT(MEL)
C
C     DEFINITIONS
C
C     JNN = # OF THE MID-SIDE NODE USED TO NUMBER THE FACE
C     NODR = NODE ID FOR RIGHT END TOP
C     NODL = NODE ID FOR LEFT END TOP
C     NQD = NUMBER OF QUADRILATERALS ALONG THIS FACE
C     NTR = NUMBER OF TRIANGLES ALONG THIS FACE
C     IDND(I,J) = NODE CONNECTION TABLE AROUND A FACE; WHERE I IS THE
C          FACE NUMBER AND J IS THE LOCAL NODE NUMBER
C     NNPE(I) = THE NUMBER OF NODES MAKING UP THIS FACE 6 OR 8; I IS
C          THE FACE NUMBER
C     NES = THE NUMBER OF SURFACE ELEMENTS (I.E. PLANAR)
C     JNM = THE NUMBER OF VERTICAL FACES
C     LIS(1-JNM)=LIST OF THE VERTICAL FACE NUMBERS
C     FLUX(I) = THE FLUX FOR FACE LIS(I)
C     VOL(I) = THE VOLUME OF ELEMENT LISE(I)
C     LISE(1-JNV) = LIST OF ELEMENTS FOR VOLUMES AND VERTICAL FLUX
C     VAREA(I) = THE VERTICAL FLUX OUT OF THE TOP OF ELEMENT I
C
C     ELEV IS HARDWIRED TO THIS APPLICATION THIS NEEDS TO BE CHANGED
C     TO SOMETHING MORE GENERAL
```

```
C


      open(unit=10,file='femconvT.out',status ='UNKNOWN')


      ELEV=100.000
C
C     READ THE GEOMETRY PORTION OF THE FILE
C
1     WRITE(*,*) ' WHAT IS THE HYDRO FILE ?'
      READ(*,'(A)',ERR=1) CHFN
      OPEN(UNIT=60,FILE=CHFN,STATUS='OLD',form='unformatted')
      READ (60) NP, NE, NPM, NES,
     &    ((CORD(J,K),SPEC(J,K),K=1,3), ALFA(J), NFIX(J), AO(J),
     &    NSURF(J), J = 1, NP), (NDEP(J), NREF(J), J = 1, NPM),
     &    ((NOP(J,K),K=1,20), NCORN(J), IMAT(J), TH(J),
     &    NFIXH(J), J = 1, NE), (WIDTH(J), J = 1, NP)
C     FIRST SETUP THE TOPOLOGY OF THE FACES
      DO 10  J = 1, NPM
        LIS(J)=0
        DO 20 I=1,8
          IDND(J,I) = 0
20       CONTINUE
10    CONTINUE
      DO 30 J=1,NP
          if(j.le.npm)then
          NNPE(J)=0
          else
          nnpe(j)=1
          endif
30    CONTINUE
```

```
C
C     LOOP THROUGH ALL THE 2D SURFACE ELEMENTS IN THE GEOMETRY
C
      jnm=0
      DO 100 IE=1,NES
        DO 110 I=1,ncorn(ie),2
          JNN = NOP(IE,I+1)
        if(nnpe(jnn).ne.0)go to 110
          JNE = (NDEP(JNN)+1)/2
          NODL=NOP(IE,I)
          IF((I+1).EQ.ncorn(ie))THEN
            NODR=NOP(IE,i-6)
          ELSE
            NODR=NOP(IE,I+2)
          END IF
          NELL = (NDEP(NODL)-1)/2
          NELR = (NDEP(NODR)-1)/2
          nqd=nell
          if(nelr.lt.nqd)nqd=nelr
          ntr=jne-nqd
          IF(NQD.NE.0) THEN
            DO 120 J=1,NQD
              IF(J.EQ.1)THEN
                IDEL = JNN
                jnm=jnm+1
                lis(jnm)=idel
                IDND(IDEL,1) = NODL
                IDND(IDEL,2) = NREF(NODL)+1
                IDND(IDEL,3) = NREF(NODL)+2
                IDND(IDEL,4) = NREF(IDEL)+1
                IDND(IDEL,5) = NREF(NODR)+2
                IDND(IDEL,6) = NREF(NODR)+1
                IDND(IDEL,7) = NODR
                IDND(IDEL,8) = JNN
```

```
            NNPE(IDEL) = 8
          ELSE
            IDEL=NREF(JNN)+J-1
            JNM=JNM+1
            LIS(JNM)=IDEL
            IDND(IDEL,1) = NREF(NODL)+(J-1)*2
            IDND(IDEL,2) = IDND(IDEL,1)+1
            IDND(IDEL,3) = IDND(IDEL,2)+1
            IDND(IDEL,4) = IDEL+1
            IDND(IDEL,7) = NREF(NODR)+(J-1)*2
            IDND(IDEL,6) = IDND(IDEL,7)+1
            IDND(IDEL,5) = IDND(IDEL,6)+1
            IDND(IDEL,8) = IDEL
            NNPE(IDEL) = 8
          END IF
120       CONTINUE
        ENDIF
C
        IF(NTR.NE.0)THEN
         DO 130 J=1, NTR
          IF (NQD.EQ.0)THEN
            IF(J.EQ.1)THEN
                IF (NELL.GT.NELR)THEN
                IDEL = JNN
                JNM=JNM+1
                LIS(JNM)=IDEL
                IDND(IDEL,1) = NODL
                IDND(IDEL,2) = NREF(NODL)+1
                IDND(IDEL,3) = NREF(NODL)+2
                IDND(IDEL,4) = NREF(IDEL)+1
                IDND(IDEL,5) = NODR
                IDND(IDEL,6) = IDEL
                NNPE(IDEL) = 6
                ELSE
```

```
            IDEL = JNN
            JNM=JNM+1
            LIS(JNM)=IDEL
            IDND(IDEL,1) = NODL
            IDND(IDEL,2) = NREF(IDEL)+1
            IDND(IDEL,5) = NODR
            IDND(IDEL,4) = NREF(NODR)+1
            IDND(IDEL,3) = NREF(NODR)+2
            IDND(IDEL,6) = IDEL
            NNPE(IDEL) = 6
            ENDIF
         ELSE
            IF(NELL.GT.NELR)THEN
            IDEL = NREF(JNN) + J - 1
            JNM=JNM+1
            LIS(JNM)=IDEL
            NOPT = NQD+J-1
            IDND(IDEL,1) = NREF(NODL)+NOPT*2
            IDND(IDEL,2) = IDND(IDEL,1)+1
            IDND(IDEL,3) = IDND(IDEL,2)+1
            IDND(IDEL,4) = IDEL+1
            IDND(IDEL,5) = NODR
            IDND(IDEL,6) = IDEL
            NNPE(IDEL) = 6
            ELSE
            IDEL = NREF(JNN) + J - 1
            JNM=JNM+1
            LIS(JNM)=IDEL
            NOPT = NQD+J-1
            IDND(IDEL,1) = NODL
            IDND(IDEL,2) = IDEL+1
            IDND(IDEL,5) = NREF(NODR) + NOPT*2
            IDND(IDEL,4) = IDND(IDEL,5)+1
            IDND(IDEL,3) = IDND(IDEL,5)+2
```

```
                IDND(IDEL,6) = IDEL
                NNPE(IDEL) = 6
                ENDIF
              ENDIF
            ELSE
              NOPT = NQD+J-1
              IDEL = NREF(JNN) + NOPT
              JNM=JNM+1
              LIS(JNM)=IDEL
              IF(NELL.GT.NELR)THEN
                IDND(IDEL,1) = NREF(NODL)+NOPT*2
                IDND(IDEL,2) = IDND(IDEL,1)+1
                IDND(IDEL,3) = IDND(IDEL,2)+1
                IDND(IDEL,4) = IDEL+1
                IDND(IDEL,5) = NREF(NODR)+NQD*2
                IDND(IDEL,6) = IDEL
                NNPE(IDEL) = 6
              ELSE
                IDND(IDEL,1) = NREF(NODL)+NQD*2
                IDND(IDEL,2) = IDEL+1
                IDND(IDEL,5) = NREF(NODR) + NOPT*2
                IDND(IDEL,4) = IDND(IDEL,5)+1
                IDND(IDEL,3) = IDND(IDEL,5)+2
                IDND(IDEL,6) = IDEL
                NNPE(IDEL) = 6
              ENDIF
            ENDIF
130         CONTINUE
          ENDIF
110       CONTINUE
100     CONTINUE
c
c       Now calculate fluxes
c
```

```
      read(60,end=200)tet,np,ndf,
     .     ((xvel(k,j),k=1,ndf),vvel(j),j=1,np),
     .     (dfct(j),j=1,ne)
C
C     THIS IS CURRENTLY SETUP FOR QUADRILATERAL ONLY
C
      DO 210 I=1,JNM
          IFACE=LIS(I)
          X1=CORD(IDND(IFACE,1),1)
          Y1=CORD(IDND(IFACE,1),2)
          Z1=CORD(IDND(IFACE,1),3)
          X3=CORD(IDND(IFACE,3),1)
          Y3=CORD(IDND(IFACE,3),2)
          Z3=CORD(IDND(IFACE,3),3)
          X5=CORD(IDND(IFACE,5),1)
          Y5=CORD(IDND(IFACE,5),2)
          Z5=CORD(IDND(IFACE,5),3)
          X7=CORD(IDND(IFACE,7),1)
          Y7=CORD(IDND(IFACE,7),2)
          Z7=CORD(IDND(IFACE,7),3)
          U1=XVEL(1,IDND(IFACE,1))
          V1=XVEL(2,IDND(IFACE,1))
          H1=XVEL(3,IDND(IFACE,1))
          U2=XVEL(1,IDND(IFACE,2))
          V2=XVEL(2,IDND(IFACE,2))
          U3=XVEL(1,IDND(IFACE,3))
          V3=XVEL(2,IDND(IFACE,3))
          H3=XVEL(3,IDND(IFACE,3))
          U4=XVEL(1,IDND(IFACE,4))
          V4=XVEL(2,IDND(IFACE,4))
          U5=XVEL(1,IDND(IFACE,5))
          V5=XVEL(2,IDND(IFACE,5))
          H5=XVEL(3,IDND(IFACE,5))
          U6=XVEL(1,IDND(IFACE,6))
```

```
      V6=XVEL(2,IDND(IFACE,6))
      U7=XVEL(1,IDND(IFACE,7))
      V7=XVEL(2,IDND(IFACE,7))
      H7=XVEL(3,IDND(IFACE,7))
      U8=XVEL(1,IDND(IFACE,8))
      V8=XVEL(2,IDND(IFACE,8))
C
C       CORRECT THE COORDINATES FOR THE TRANSFORMATION
C
      Z1=(Z1-AO(IDND(IFACE,1)))/(ELEV-AO(IDND(IFACE,1)))
     .     *XVEL(3,IDND(IFACE,1))+AO(IDND(IFACE,1))
      Z3=(Z3-AO(IDND(IFACE,3)))/(ELEV-AO(IDND(IFACE,3)))
     .     *XVEL(3,IDND(IFACE,3))+AO(IDND(IFACE,3))
      Z5=(Z5-AO(IDND(IFACE,5)))/(ELEV-AO(IDND(IFACE,5)))
     .     *XVEL(3,IDND(IFACE,5))+AO(IDND(IFACE,5))
      Z7=(Z7-AO(IDND(IFACE,7)))/(ELEV-AO(IDND(IFACE,7)))
     .     *XVEL(3,IDND(IFACE,7))+AO(IDND(IFACE,7))
C
C       CALCULATE HORIZONTAL FLUXES
C
C       reference distances to a corner
      X1 = X1 - X7
      Y1 = Y1 - Y7
      Z1 = Z1 - Z7
      X3 = X3 - X7
      Y3 = Y3 - Y7
      Z3 = Z3 - Z7
      X5 = X5 - X7
      Y5 = Y5 - Y7
      Z5 = Z5 - Z7
C
C       temporary variables for jacobians
      X1Z1 = X1 * Z1
      X3Z1 = X3 * Z1
```

```
      X5Z1 = X5 * Z1
      X1Z3 = X1 * Z3
      X3Z3 = X3 * Z3
      X5Z3 = X5 * Z3
      X1Z5 = X1 * Z5
      X3Z5 = X3 * Z5
      X5Z5 = X5 * Z5
C
      Y1Z1 = Y1 * Z1
      Y3Z1 = Y3 * Z1
      Y5Z1 = Y5 * Z1
      Y1Z3 = Y1 * Z3
      Y3Z3 = Y3 * Z3
      Y5Z3 = Y5 * Z3
      Y1Z5 = Y1 * Z5
      Y3Z5 = Y3 * Z5
      Y5Z5 = Y5 * Z5
C
      FX = (U1*(-Y1Z3+Y3Z1-2.*Y3Z5+2.*Y5Z3)
     .     +2.*U2*(-4.*Y3Z1+Y5Z1+4.*Y1Z3
     .          -3.*Y5Z3-Y1Z5+3.*Y3Z5)
     .     +U3*(Y3Z1+Y5Z1-Y1Z3+Y5Z3-Y1Z5-Y3Z5)
     .     +2.*U4*(-3.*Y3Z1+Y5Z1+3.*Y1Z3
     .          -4.*Y5Z3-Y1Z5+4.*Y3Z5)
     .     +U5*(-2.*Y1Z3+2.*Y3Z1+Y5Z3-Y3Z5)
     .     +2.*U6*(2.*Y1Z3-2.*Y3Z1-Y5Z1
     .          -3.*Y5Z3+Y1Z5+3.*Y3Z5)
     .     +U7*(-2.*Y1Z3-Y5Z1+Y1Z5+2.*Y3Z1
     .          -2.*Y3Z5+2.*Y5Z3)
     .     +2.*U8*(-3.*Y3Z1-Y5Z1+3.*Y1Z3
     .          +Y1Z5+2.*Y3Z5-2.*Y5Z3))
     .     /36.
C
```

```
          FY = (V1*(-X1Z3+X3Z1-2.*X3Z5+2.*X5Z3)
     .          +2.*V2*(-4.*X3Z1+X5Z1+4.*X1Z3
     .               -3.*X5Z3-X1Z5+3.*X3Z5)
     .          +V3*(X3Z1+X5Z1-X1Z3+X5Z3-X1Z5-X3Z5)
     .          +2.*V4*(-3.*X3Z1+X5Z1+3.*X1Z3
     .               -4.*X5Z3-X1Z5+4.*X3Z5)
     .          +V5*(-2.*X1Z3+2.*X3Z1+X5Z3-X3Z5)
     .          +2.*V6*(2.*X1Z3-2.*X3Z1-X5Z1
     .               -3.*X5Z3+X1Z5+3.*X3Z5)
     .          +V7*(-2.*X1Z3-X5Z1+X1Z5+2.*X3Z1
     .               -2.*X3Z5+2.*X5Z3)
     .          +2.*V8*(-3.*X3Z1-X5Z1+3.*X1Z3
     .               +X1Z5+2.*X3Z5-2.*X5Z3))
     .          /36.
          FLUX(I) = FX + FY
          AX = 0.5*(Y1Z3-Y3Z1+Y3Z5-Y5Z3)
          AY = 0.5*(X1Z3-X3Z1+X3Z5-X5Z3)
          AREA(I)=SQRT(AX*AX+AY*AY)
          if((lis(i).gt.980).and.(lis(i).lt.990))then
          WRITE(*,*) 'FX,FY', FX,FY
          WRITE(*,*)LIS(I),FLUX(I),AREA(I)
          endif
210     CONTINUE
200     continue
C
C     3D ELEMENTS VOLUME CALCULATION
C
      JNV = 0
      DO 300 IE=1,NE
          IF (NCORN(IE).EQ.20)THEN
              JNV = JNV+1
              LISE(JNV)=IE
              IB1=NOP(IE,1)
              IB3=NOP(IE,3)
```

```
      IB5=NOP(IE,5)
      IB7=NOP(IE,7)
      IT1=NOP(IE,13)
      IT2=NOP(IE,14)
      IT3=NOP(IE,15)
      IT4=NOP(IE,16)
      IT5=NOP(IE,17)
      IT6=NOP(IE,18)
      IT7=NOP(IE,19)
      IT8=NOP(IE,20)
      if(ie.eq.569)then
          write(*,*)'ib',ib1,ib3,ib5,ib7
          write(*,*)'it',it1,it3,it5,it7
      end if
      X1=CORD(IB1,1)
      X3=CORD(IB3,1)
      X5=CORD(IB5,1)
      X7=CORD(IB7,1)
      Y1=CORD(IB1,2)
      Y3=CORD(IB3,2)
      Y5=CORD(IB5,2)
      Y7=CORD(IB7,2)
      ZB1=CORD(IB1,3)
      ZB3=CORD(IB3,3)
      ZB5=CORD(IB5,3)
      ZB7=CORD(IB7,3)
      ZT1=CORD(IT1,3)
      ZT3=CORD(IT3,3)
      ZT5=CORD(IT5,3)
      ZT7=CORD(IT7,3)
      ZB1=(ZB1-AO(IB1))/(ELEV-AO(IB1))
     *XVEL(3,IB1)+AO(IB1)
      ZB3=(ZB3-AO(IB3))/(ELEV-AO(IB3))
```

```
     .          *XVEL(3,IB3)+AO(IB3)
               ZB5=(ZB5-AO(IB5))/(ELEV-AO(IB5))
     .          *XVEL(3,IB5)+AO(IB5)
               ZB7=(ZB3-AO(IB7))/(ELEV-AO(IB7))
     .          *XVEL(3,IB7)+AO(IB7)
               ZT1=(ZT1-AO(IT1))/(ELEV-AO(IT1))
     .          *XVEL(3,IT1)+AO(IT1)
               ZT3=(ZT3-AO(IT3))/(ELEV-AO(IT3))
     .          *XVEL(3,IT3)+AO(IT3)
               ZT5=(ZT5-AO(IT5))/(ELEV-AO(IT5))
     .          *XVEL(3,IT5)+AO(IT5)
               ZT7=(ZT3-AO(IT7))/(ELEV-AO(IT7))
     .          *XVEL(3,IT7)+AO(IT7)
               X3Y1=X3*Y1
               X5Y1=X5*Y1
               X7Y1=X7*Y1
               X1Y3=X1*Y3
               X5Y3=X5*Y3
               X7Y3=X7*Y3
               X1Y5=X1*Y5
               X3Y5=X3*Y5
               X7Y5=X7*Y5
               X1Y7=X1*Y7
               X3Y7=X3*Y7
               X5Y7=X5*Y7
C
C               CALCULATE THE VOLUME OF AN ELEMENT
C
               VOL(JNV)=(ZB1-ZT1)*(2.*X3Y1-2.*X7Y1-2.*X1Y3+X5Y3
     .              +X7Y3-X3Y5+X7Y5+2.*X1Y7-X3Y7-X5Y7)/12.
     .              +(ZB3-ZT3)*(2.*X3Y1-X5Y1-X7Y1-2.*X1Y3+
     .              2.*X5Y3+X1Y5-2.*X3Y5+X7Y5+X1Y7-X5Y7)/12.
     .              +(ZB5-ZT5)*(X3Y1-X7Y1-X1Y3+2.*X5Y3-X7Y3
     .              -2.*X3Y5+2.*X7Y5+X1Y7+X3Y7-2.*X5Y7)/12.
```

```
     .          +(ZB7-ZT7)*(X3Y1+X5Y1-2.*X7Y1-X1Y3+X5Y3
     .          -X1Y5-X3Y5+2.*X7Y5+2.*X1Y7-2.*X5Y7)/12.
C
C          CALCULATE THE VERTICAL FLUX AND AREA
C
           W1=VVEL(IT1)
           W2=VVEL(IT2)
           W3=VVEL(IT3)
           W4=VVEL(IT4)
           W5=VVEL(IT5)
           W6=VVEL(IT6)
           W7=VVEL(IT7)
           W8=VVEL(IT8)
           VAREA(JNV)=0.5*(-X3Y1+X7Y1+X1Y3-X5Y3-X7Y5-X1Y7
     .          +X5Y7+X3Y5)
           TMP1=W1*((X3Y1-X7Y1-X1Y3+X5Y3-X3Y5+X7Y5+X1Y7-
     .          X5Y7)/12.+(-2.*X3Y1+2.*X7Y1+2.*X1Y3-X5Y3-
     .          X7Y3+X3Y5-X7Y5-2.*X1Y7+X3Y7+X5Y7)/36.)
           TMP2=W2*((4.*X3Y1-X5Y1-3.*X7Y1-4.*X1Y3+3.*X5Y3+
     .          X7Y3+X1Y5-3.*X3Y5+2.*X7Y5+3.*X1Y7-
     .          X3Y7-2.*X5Y7)/36.+
     .          (-4.*X3Y1+X5Y1+3.*X7Y1+4.*X1Y3-3.*X5Y3-
     .          X7Y3-X1Y5+3.*X3Y5-2.*X7Y5-3.*X1Y7+
     .          X3Y7+2.*X5Y7)/12.)
           TMP3=W3*((X3Y1-X7Y1-X1Y3+X5Y3-X3Y5+X7Y5+
     .          X1Y7-X5Y7)/12.+
     .          (-2.*X3Y1+X5Y1+X7Y1+2.*X1Y3-2.*X5Y3-X1Y5+
     .          2.*X3Y5-X7Y5-X1Y7+X5Y7)/36.)
           TMP4=W4*((X5Y1-X7Y1-X5Y3+X7Y3-X1Y5+X3Y5+
     .          X1Y7-X3Y7)/18.+
     .          (-X3Y1+X7Y1+X1Y3-X5Y3+X3Y5-X7Y5-
     .          X1Y7+X5Y7)/6.)
           TMP5=W5*((X3Y1-X7Y1-X1Y3+X5Y3-X3Y5+X7Y5+
     .          X1Y7-X5Y7)/12.+
```

```
     .              (-X3Y1+X7Y1+X1Y3-2.*X5Y3+X7Y3+2.*X3Y5-
     .              2.*X7Y5-X1Y7-X3Y7+2.*X5Y7)/36.)
                TMP6=W6*((2.*X3Y1+X5Y1-3.*X7Y1-2.*X1Y3+3.*X5Y3-
     .              X7Y3-X1Y5-3.*X3Y5+4.*X7Y5+3.*X1Y7+
     .              X3Y7-4.*X5Y7)/36.+
     .              (-2.*X3Y1-X5Y1+3.*X7Y1+2.*X1Y3-3.*X5Y3+
     .              X7Y3+X1Y5+3.*X3Y5-4.*X7Y5-3.*X1Y7-
     .              X3Y7+4.*X5Y7)/12.)
                TMP7=W7*((X3Y1-X7Y1-X1Y3+X5Y3-X3Y5+X7Y5+
     .              X1Y7-X5Y7)/12.+
     .              (-X3Y1-X5Y1+2.*X7Y1+X1Y3-X5Y3+X1Y5+
     .              X3Y5-2.*X7Y5-2.*X1Y7+2.*X5Y7)/36.)
                TMP8=W8*((-X5Y1+X7Y1+X5Y3-X7Y3+X1Y5-X3Y5-
     .              X1Y7+X3Y7)/18.+
     .              (-X3Y1+X7Y1+X1Y3-X5Y3+X3Y5-X7Y5-
     .              X1Y7+X5Y7)/6.)
                VFLUX(JNV)=TMP1+TMP2+TMP3
     .              +TMP4+TMP5+TMP6+TMP7+TMP8
             ENDIF
300       CONTINUE
C
C       THE VARIABLES YOU WANT OUT OF THIS ARE

          write(10,5000) JNM

          DO  I=1,JNM
              J=LIS(I)
              WRITE(10,1000) J,AREA(I),FLUX(I)        ! FACE,AREA OF FACE, FLUX
          END DO

          write(10,5000) JNV

          DO I=1,JNV
              J=LISE(I)
```

```
      WRITE(10,2000) I,VOL(I),VAREA(I),VFLUX(I) ! ELEMENT,VOLUME,AREA
                                    ! OF TOP, VERTICAL
                                    ! FLUX OUT TOP
      END DO

      close(10)

5000  format(i8)
1000  format(i8,2x,e15.9,2x,e15.9)
2000  format(i8,2x,e15.9,2x,e15.9,2x,e15.9)

      END
```

Appendix C
Modifications to ICM Code

Several modifications were made to the original ICM source code to allow it to be driven by the RMA10 hydrodynamic model. The changes made are detailed below:

Modification 1:
The inline code for reading the map file:

```
***** Flow mapping data

    OPEN (MAP,FILE=MAPFN,STATUS='OLD')
    READ (MAP,1110) (QD(F),ILB(F),IB(F),JB(F),JRB(F),F=1,NQF)
    READ (MAP,1027) (NVF(SB),SB=1,NSB)
    READ (MAP,1100)
    DO 10010 SB=1,NSB
     READ (MAP,1130) (VFN(F,SB),F=1,NVF(SB))
10010 CONTINUE
    READ (MAP,1027) (OBP(F),F=1,NOBP)
    CLOSE (MAP)
```

was changed and implemented as a subroutine:

```
    subroutine readMap(MAPFN)

    character*72 MAPFN

    include  'model.inc'


    integer  HHtag(NFEMHFFP), HVtag(NFEMVFFP), Itag(NQFP),
   .       HHtagToItagMap(NFEMHFFP), HVtagToItagMap(NFEMVFFP),
   .       FemCellToIcmCell(NBP)
    common   /lookup/ HHtag, HVtag, Itag, HHtagToItagMap,
   .            HVtagToItagMap, FemCellToIcmCell
```

```
      integer  F, SB, ISB
      integer  icmCellId, femCellId

      open(MAP,FILE=MAPFN,STATUS='OLD')

      read(map,1000)
      do b=1,NBP
       read(map,1500) icmCellId, femCellId
       FemCellToIcmCell(femCellId) = icmCellId
      enddo

      read(map,1000)
      read(MAP,2000) (ILB(F),IB(F),JB(F),JRB(F),ITAG(F),F=1,NQF)

      read(map,1000)
      read(MAP,2010) (NVF(SB),SB=1,NSB)

      read(map,1000)
      do SB=1,NSB
       READ (MAP,2030) ISB, (VFN(F,SB),F=1,NVF(SB))
      enddo

      close(MAP)

      return

 1000 format(/)
 1500 format(I8,I8)
 2000 format(8X,4I8,15x,I6)
 2010 format(8X,I8)
 2020 format(/)
 2030 format(I8,9I8)

      end
```

Modification 2:
The inline code for reading the geo file:

```
***** Geometric data

      OPEN (GEO,FILE=GEOFN,STATUS='OLD')
      IF (BINARY_HYDRO .OR. DEPTH_AVG_HYDRO) THEN
       READ (GEO,1120) (BU(B),B=1,NB)
       READ (GEO,1140) (SBN(SB),BBN(SB),SB=1,NSB)
      ELSE
       READ (GEO,1000)
       READ (GEO,1150) (BL(B,1),BL(B,2),BL(B,3),V1(B),ZD(B),BU(B),
     .            B=1,NB)
       READ (GEO,1170) (SBN(SB),BBN(SB),SB=1,NSB)
       DO 10015 SB=1,NSB
        SFA(SB) = V1(SBN(SB))/BL(SBN(SB),3)
10015  CONTINUE
      END IF
      CLOSE (GEO)
```

was changed and implemented as a subroutine:

```
      subroutine readGeo(GEOFN)
      character*72 GEOFN

      include  'model.inc'

      integer  f,sb

      dimension BIL(NHQP), BI(NHQP), BID(NHQP), BIR(NHQP)
      common   /rma10/ BIL, BI, BID, BIR
```

```
c                                   Data from map generator
   open (geo,file=GEOFN,status='OLD')

    read(geo,1000)
c                                   Surface box id
    read(geo,1020) (SBN(SB),BBN(SB),SB=1,NSB)

    read(geo,1040)
c                                   Upper box for each box
    read(geo,1025) (bu(b),b=1,nb)

    read(geo,1040)
c                                   Centroid distances
    do f=1,NHQP
     read(geo,1010) BIL(f), BI(f), BID(f), BIR(f)
    enddo

c                                   NOTE !! following moved
c                                   to hydro file.
c                                   Box urface area
c    read(geo,1030) (sfa(sb),sb=1,nsb)

    close (geo)

    return

1000  format(1x)
1010  format(4(2x,e11.6))
1020  format(I6,I7)
1025  format(I6)
1030  format(e15.9)
1040  format(/)

    end
```

Modification 3:

The inline code for reading the time-invariant hydro data from the hydro file:

```
***** Time-invariant hydrodynamic data

    IF (BINARY_HYDRO) THEN
      READ (HYD)  SFA
      READ (HYD) (BL(SB,1),SB=1,NSB)
      READ (HYD) (BL(SB,2),SB=1,NSB)
      READ (HYD) (A(F),F=1,NHQF)
      READ (HYD)  HMBV
      READ (HYD)  HMSBV
    ELSE IF (ASCII_HYDRO) THEN
      READ (HYD,1000)
      READ (HYD,1160) (A(F),F=1,NQF)
      READ (HYD,1100)
    ELSE IF (DEPTH_AVG_HYDRO) THEN
      READ (HYD)  SFA
      READ (HYD) (A(F),F=1,NHQF)
      READ (HYD) (BL(SB,1),SB=1,NSB)
      READ (HYD) (BL(SB,2),SB=1,NSB)
      READ (HYD)  HMSBV
    ELSE
      WRITE(*,*) 'hydro file specified incorrectly'
      STOP
    ENDIF
```

was changed and implemented as a subroutine:

```
    logical function initializeHydro()
    include  'model.inc'
```

```
      integer  HHtag(NFEMHFFP), HVtag(NFEMVFFP), Itag(NQFP),
     .         HHtagToItagMap(NFEMHFFP), HVtagToItagMap(NFEMVFFP),
     .         FemCellToIcmCell(NBP)

      common   /lookup/ HHtag, HVtag, Itag, HHtagToItagMap,
     .          HVtagToItagMap, FemCellToIcmCell

      integer  HYD2
      common   /lookup2/ HYD2

      integer  node_id, hface_id, vface_id, box_id
      real     f_area, flux,  volume,  top_area, v_flux

      integer  i, f1, f2, b1

      integer  nHHtags1, nHVTags2, HHtagValue, HVtagValue,
     .         ItagIndex

      integer  unusedHFaceCount, unusedVFaceCount

      logical  end_of_file, readHydro


c                        face_id: id of flow face
c                        f_area: Flow face area
c                  flux: Flow face flux
c                        box_id: id of box
c                        volume: box volume
c                        top_area: area of box top
c                        v_flux: flux out of boxes top face

c                        Clear the following arrays
      do i=1,NFEMHFFP
       HHtagToItagMap(i) = 0
```

```
      enddo

      do i=1,NFEMVFFP
       HVtagToItagMap(i) = 0
      enddo


      open(HYD2,FILE=HYDFN(1),STATUS='OLD',FORM = 'FORMATTED')

      initializeHydro = .TRUE.

c                       Read in horizontal flow face
c                       data from rma10 output file

      read(hyd2,100,end=50) nHHtags1

      f2 = 0
      do f1=1,nHHtags1
       read(hyd2,200,end=50) node_id, f_area, flux
       f2 = f2 + 1
       HHtag(f2) = node_id

      enddo

      unusedHFaceCount = 0

      do i = 1, f2

       HHtagValue = HHtag(i)
       ItagIndex = 0
       j = 0

20     continue
```

```
      j = j + 1
      if(j .gt. NHQP) goto 30

      if(Itag(j) .eq. HHtagValue) then
 ItagIndex = j
 goto 30
else
 goto 20
endif

 30    continue

 c                        If HtagToItagMap(HtagValue) = 0 then
 c                        not a "flow face" we are interested in!

      HHtagToItagMap(i) = ItagIndex
   if(ItagIndex .eq. 0)  unusedHFaceCount  = unusedHFaceCount+1

    enddo

 c                        Read in cell data and vertical
 c                        flow face data
    read(hyd2,100,end=50) nHVtags2

    f2 = 0
    do b1=1,nHVtags2
     read(hyd2,300) box_id, volume, top_area, v_flux
     f2 = f2 + 1
     HVtag(f2) = box_id
    enddo

    close(HYD2)
```

```
      unusedVFaceCount = 0

      do i = 1, f2

       HVtagValue = HVtag(i)
       ItagIndex = 0
       j = NHQP-1

2000   continue

       j = j + 1
       if(j .gt. NQFP) goto 3000

       if(Itag(j) .eq. HVtagValue) then
        ItagIndex = j
        goto 3000
       else
        if(-Itag(j) .eq. HVtagValue) then
         ItagIndex = -j
        else
         goto 2000
        endif
       endif

3000   continue

c                      If HtagToItagMap(HtagValue) = 0 then
c                      not a "flow face" we are interested in!

      HVtagToItagMap(HVtagValue) = ItagIndex
      if(ItagIndex .eq. 0)  unusedVFaceCount  = unusedVFaceCount+1

     enddo
```

```
      write(*,*) ' finished tag mapping'

      close(HYD2)

c                       Need to initialize HMV (Hydrodynamic
c                       model cell volume) array and A (ICM flow
c                       face area) array. Only way to do this is
c                       to read data in from hydro file.

      open(HYD,FILE=HYDFN(1),STATUS='OLD',FORM = 'FORMATTED')

      end_of_file = readHydro()
      if(end_of_file) then
       initializeHydro = .TRUE.
      else
       initializeHydro = .FALSE.
      endif

      close(2)

      return

50    close(HYD)

      return

100   format(i8)
200   format(I8,2x,e15.9,2x,e15.9)
300   format(I8,2x,e15.9,2x,e15.9,2x,e15.9)

      end
```

Modification 4:
A subroutine for reading the time-variant hydro data was written to read the RMA10 hydro output data. This routine replaces ICM's HYDRO subroutine. New subroutine code follows:

```
      logical function readHydro()
      include 'model.inc'

      integer  HHtag(NFEMHFFP), HVtag(NFEMVFFP), Itag(NQFP),
     .         HHtagToItagMap(NFEMHFFP), HVtagToItagMap(NFEMVFFP),
     .         FemCellToIcmCell(NBP)

      common   /lookup/ HHtag, HVtag, Itag, HHtagToItagMap,
     .                  HVtagToItagMap, FemCellToIcmCell

      logical  firstHR
      common   /lookup3/firstHR
      logical  end_of_file
      integer  node_id, hface_id, vface_id, box_id
      real     f_area, flux, volume, top_area, v_flux
      integer  unusedHFaceCount, unusedVFaceCount
      real     sense
      integer  icm_box_id, fem_box_id
      integer  f
      logical  bTagMap

c                NOTE the PARAMETER NFEMFFP MEANS ...
c                Number of Finite Element Model Flow
c                Faces Parameter
c                NFEMFFP = nHtags1 + nHtags2


c                                face_id: id of flow face
c                           f_area: Flow face area
```

```
c                flux: Flow face flux
c                      icm_box_id: id of box in ICM
c                      fem_box_id: id of box in RMA0
c                      volume: box volume
c                      top_area: area of box top
c                      v_flux: flux out of boxes top face


     readHydro = .FALSE.
c                      Read in horizontal flow face
c                      data from rma10 output file

     unusedHFaceCount = 0
     read(hyd,100,end=50) nHtags1
     do f=1,nHtags1

      read(hyd,200,end=50) node_id, f_area, flux

      hface_id = HHtagToItagMap(f)

      if(hface_id .eq. 0) then
       unusedHFaceCount = unusedHFaceCount + 1
      else
       a(hface_id) = f_area
       q(hface_id) = flux
      endif

     enddo

c                      Read in cell data and vertical
c                      flow face data

     unusedVFaceCount = 0
     read(hyd,100,end=50) nHtags2
```

```
      do b=1,nHtags2
       read(hyd,300) fem_box_id, volume, top_area, v_flux
       icm_box_id = FemCellToIcmCell(fem_box_id)
       hmv(icm_box_id) = volume
       bl3(icm_box_id) = volume/top_area
       sfa(icm_box_id) = top_area
       vface_id = HVtagToItagMap(fem_box_id)
       if(vface_id .eq. 0) then
        unusedVFaceCount = unusedVFaceCount + 1
       else
        q(vface_id) = v_flux
        a(vface_id) = top_area
       endif
      enddo
      return

50    readHydro = .TRUE.
      return

100   format(i8)
200   format(I8,2x,e15.9,2x,e15.9)
300   format(I8,2x,e15.9,2x,e15.9,2x,e15.9)

      end
```

Modification 5:

The BL array was originally implemented as a two-dimensional array: BL(NUMBER_OF_BOXES,3). This array holds the three lengths (width, length and height) for each cell. The finite-element version does not distingush between the horizontal distances; therefore this array was replaced with the array BL3(NUMBER_OF_BOXES). This array contains the cell height. All original code references were changed to reflect this modification.

Modification 6:
The mainline initialization of the HMV and V1 arrays:

```
********* Initialize volumes

 DO 10110 SB=1,NSB
  DO 10100 F=1,NVF(SB)
   HMV(IB(VFN(F,SB))) = HMBV(SB)
   V1(IB(VFN(F,SB)))  = HMBV(SB)
10100    CONTINUE
  HMV(JB(VFN(NVF(SB),SB))) = HMSBV(SB)
  V1(JB(VFN(NVF(SB),SB)))  = HMSBV(SB)
10110  CONTINUE
```

was replaced with the following code:

```
********* Initialize volumes

 do 10100 b=1,NBP
  v1(b) = hmv(b)
10100  continue
```

The FEM verison of ICM reads in from the hydrodynamic input file the volume of each box and assigns it directly to the array HMV. There is no intermediate assignment to HMBV (volume of boxes below surface layer with the index being the surface box id) or to HMSBV (volume of boxes in surface layer).

Modification 7:
The mainline initialization of the BL array was eliminated:

```
********* Initialize box lengths

      DO 10130 SB=1,NSB
CDIR$ VECTOR
       DO 10120 F=1,NVF(SB)
        BL(IB(VFN(F,SB)),1) = BL(JB(VFN(NVF(SB),SB)),1)
        BL(IB(VFN(F,SB)),2) = BL(JB(VFN(NVF(SB),SB)),2)
        BL(IB(VFN(F,SB)),3) = HMV(IB(VFN(F,SB)))/SFA(SB)
10120    CONTINUE
       BL(JB(VFN(NVF(SB),SB)),3) = HMV(JB(VFN(NVF(SB),SB)))/SFA(SB)
10130   CONTINUE
```

The Bl array was replaced with the BL3 array. The BL3 array is assigned by the subroutine READHYDRO and updated by the subroutine UPDATE.

Modification 8:
Subroutine calls made to HYDRO changed to calls of HYDRO2. The original subroutine HYDRO was not modified. Instead, a new routine specific to RMA10 was written. This new routine handles the hydrodynamic data updates.

```
SUBROUTINE HYDRO2 (NXHYD)

SAVE

INCLUDE 'model.inc'


REAL     NXDAY
INTEGER  F,    SB
LOGICAL  END_OF_FILE, readHydro
DIMENSION MASS(0:NBP,NCP)
```

```
******* Binary time-varying hydrodynamic data


10000  IF (NWQMR.GE.NHMR) THEN

*****                              Hydrodynamic column volumes, flows,
*****                              and vertical diffusions

      end_of_file = readHydro()

10020   IF (END_OF_FILE) THEN

*****                              Open next hydrodynamic file

      HYDPTR = HYDPTR+1
      IF (DIAGNOSTICS) WRITE (DIA,*) 'Opening hydrodynamic ',
     .                        'file ',HYDPTR,' at day ',
     .                        JDAY
      CLOSE (HYD)
      OPEN  (HYD,FILE=HYDFN(HYDPTR),FORM='UNFORMATTED',
     .       STATUS='OLD')

      end_of_file = readHydro()

      NWQMR = 0
      NHMR  = 0

*****                                        Reinitialize HM and WQM volumes

      do b=1,NBP
       v2(b) = hmv(b)
      enddo

      IF (CONSERVE_MASS) THEN
```

```
            DO 10047 B=1,NB
             DO 10045 JC=1,NAC
              MASS(B,AC(JC)) = C1(B,AC(JC))*V1(B)
10045          CONTINUE
             V1(B)  = V2(B)
             V1S(B) = V1(B)
             DO 10046 JC=1,NAC
              C1(B,AC(JC)) = MASS(B,AC(JC))/V1(B)
              C2(B,AC(JC)) = MAX(C1(B,AC(JC)),0.0)
10046         CONTINUE
10047        CONTINUE
           ELSE
            DO 10048 B=1,NB
             V1(B) = V2(B)
10048        CONTINUE
           END IF

*****                                  Reinitialize initial water column mass

           IF (MASS_BALANCE) THEN

*****                                  Reinitialize mass balance variables

            S1FLXN    = 0.
            S1FLXP    = 0.
            S1FLXC    = 0.
            S2FLXN    = 0.
            S2FLXP    = 0.
            S2FLXC    = 0.
            ATMFLXN   = 0.
            ATMFLXP   = 0.
            BENFLXPN  = 0.
            BENFLXDN  = 0.
            BENFLXPP  = 0.
```

```
      BENFLXDP  = 0.
      BENFLXPC  = 0.
      DLWCKMN   = 0.
      DLWCKMC   = 0.
      BNDFLXN   = 0.
      BNDFLXP   = 0.
      BNDFLXC   = 0.
      BURIALFLXN = 0.
      BURIALFLXP = 0.
      BURIALFLXC = 0.
      DLSEDKN   = 0.
      DLSEDKC   = 0.
      ISEDMN    = 0.
      ISEDMP    = 0.
      ISEDMC    = 0.

*****                          Reinitialize initial water column and
*****                          sediment masses

      DO 10051 J=1,NAC
       JC      = AC(J)
       CMASS(JC) = 0.0
       DO 10050 B=1,NB
        CMASS(JC) = CMASS(JC)+C1(B,JC)*V1(B)/1000.
10050    CONTINUE
10051   CONTINUE
      IWCMN = ANCC*CMASS(4)+ANCD*CMASS(5)+ANCG*CMASS(6)
     .       +CMASS(10)+CMASS(11)+CMASS(12)+CMASS(13)
     .       +CMASS(14)
      IWCMP = CMASS(15)+CMASS(16)+CMASS(17)+CMASS(18)
      IWCMC = CMASS(4)+CMASS(5)+CMASS(6)+CMASS(7)+CMASS(8)
     .       +CMASS(9)
      IWCMS = ASCD*CMASS(5)+CMASS(21)+CMASS(22)
      DO 10058 BB=1,NBB
```

```
            ISEDMN = ISEDMN+(CPON(BB,1)+CPON(BB,2)+CPON(BB,3)
     .                +CNH4(BB)+CNO3(BB))*A(VFN(1,BB))
     .                *HSED(BB)/1.E6
            ISEDMP = ISEDMP+(CPOP(BB,1)+CPOP(BB,2)+CPOP(BB,3)
     .                +CPIP(BB))*A(VFN(1,BB))*HSED(BB)/1.E6
            ISEDMC = ISEDMC+(CPOC(BB,1)+CPOC(BB,2)+CPOC(BB,3))
     .                *A(VFN(1,BB))*HSED(BB)/1.E6
10058       CONTINUE
          END IF

          END_OF_FILE = .FALSE.

        ELSE

*****                                  Reinitialize HM and WQM volumes
          do b=1,NBP
           v2(b) = hmv(b)
          enddo

          NHMR = NHMR+1

        END IF

        GO TO 10000
      END IF
      NWQMR = NWQMR+1
      NXHYD = NXHYD+INT(AHMDLT)


*******              Calculations


******* Dead sea case
```

```
      IF (.NOT.FLOW) THEN
       DO 10100 F=1,NQF
        Q(F) = 0.0
10100    CONTINUE
      END IF
      IF (.NOT.XY_DIFFUSION) THEN
       DO 10110 F=1,NHQF
        DIFF(F) = 0.
10110    CONTINUE
      END IF

      IF (.NOT.Z_DIFFUSION) THEN
       DO 10120 F=NHQF+1,NQF
        DIFF(F) = 0.
10120    CONTINUE
      END IF


******* Determine flow direction

      DO 10130 F=1,NQF
       IF (Q(F).GE.0.0) THEN
        POSITIVE_FLOW(F) = .TRUE.
       ELSE
        POSITIVE_FLOW(F) = .FALSE.
       END IF
10130  CONTINUE

******* Adjust vertical diffusion


      DO 10140 F=NHQF+1,NQF
       DIFF(F) = ZDFMUL*DIFF(F)
10140  CONTINUE
```

```
******* ASCII input FORMAT statements

 1000  FORMAT(21X,E10.3,5X,E10.3)
 1005  FORMAT(F8.0)
 1010  FORMAT(///)
 1020  FORMAT(13X,F13.0)
 1030  FORMAT(/)

      END
```

Modification 9:
The following mainline code for updating the geometry variables:

```
********* Update WQM volumes

      DO 10600 F=1,NQF

       V2(JB(F)) = V2(JB(F))+Q(F)*DLT
       V2(IB(F)) = V2(IB(F))-Q(F)*DLT

10600   CONTINUE

********* Update box lengths

      DO 10620 SB=1,NSB
       DO 10610 F=1,NVF(SB)
        BL(IB(VFN(F,SB)),3) = V2(IB(VFN(F,SB)))/SFA(SB)
10610    CONTINUE
       BL(JB(VFN(NVF(SB),SB)),3) = V2(JB(VFN(NVF(SB),SB)))
                      /SFA(SB)
10620   CONTINUE
```

```
********* Update layer depths

      DO 10630 B=1,NB
       ZD(B) = 0.0
10630   CONTINUE
      DO 10650 SB=1,NSB
CDIR$ VECTOR
       DO 10640 F=NVF(SB),1,-1
        ZD(IB(VFN(F,SB))) = ZD(JB(VFN(F,SB)))
     .                    +BL(JB(VFN(F,SB)),3)
10640    CONTINUE
10650   CONTINUE

     ELSE IF (DEPTH_AVG_HYDRO) THEN

********* Update WQM volumes

       DO 50600 F=1,NQF

        V2(JB(F)) = V2(JB(F))+Q(F)*DLT
        V2(IB(F)) = V2(IB(F))-Q(F)*DLT

50600    CONTINUE

********* Update box lengths

       DO 50620 SB=1,NSB
         BL(SB,3) = V2(SB) / SFA(SB)
50620    CONTINUE
      END IF
```

was replaced with a call to the new subroutine UPDATE:

```
       subroutine updateG(HYDC)
```

```
      character*8 HYDC
      integer    f,sb

      include    'model.inc'
      integer    f, b, sb

c                          Check type of hydrodynamics
      if(HYDC.EQ.' BINARY') goto 10
      if(HYDC.EQ.'DEPTH_AV') goto 20
      if(HYDC.EQ.'RMAHYDRO') goto 30

c                          BINARY_HYDRO
10    continue

c                          Update WQM volumes
      do f=1,NQF
       V2(JB(f)) = V2(JB(f))+Q(f)*DLT
       V2(IB(f)) = V2(IB(f))-Q(f)*DLT
      enddo
c                                  Update box lengths..Note that
c                                  vertical box length changes
c                                  every time step because box
c                                  volume is changing every time
c                                  step!

      do b=1,nb
       bl3(b) = v2(b)/sfa(b)
      enddo
c                                  Update layer depths

c                                  ZD(i) is depth below surface
c                                  of top of box i!
c                                  cell depth below equals
c                                  cell depth above + thickness
```

```
c                                   of cell above


      do b=1,NB
        ZD(b) = 0.0
      enddo

      do sb=1,NSB
       do f=NVF(sb),1,-1
        ZD(IB(VFN(f,sb))) = ZD(JB(VFN(f,sb)))
                          +BL3(JB(VFN(f,sb)))
       enddo
      enddo

      return


c                                   DEPTH_AVG_HYDRO
20    continue

c                                   Set depth to box top
      do b=1,NB
        ZD(b) = 0.0
      enddo
c                                   Update WQM volumes
      do f=1,NQF
       V2(JB(f)) = V2(JB(f))+Q(f)*DLT
       V2(IB(f)) = V2(IB(f))-Q(f)*DLT
      enddo

c                                   Update box lengths
      return

c                                   RMA10_HYDRO
```

```
30   continue

c                                   Update WQM volumes
     do  f=1,NQF
      V2(JB(f)) = V2(JB(f))+Q(f)*DLT
      V2(IB(f)) = V2(IB(f))-Q(f)*DLT
     enddo

c                                   Update box lengths
     do b=1,nb
      bl3(b) = v2(b)/sfa(b)
     enddo

c                                   Set depth to top of box

     do b=1,NB
       ZD(b) = 0.0
     enddo

     do sb=1,NSB
      do f=NVF(sb),1,-1
       ZD(IB(VFN(f,sb))) = ZD(JB(VFN(f,sb)))
                         +BL3(JB(VFN(f,sb)))
      enddo
     enddo

    return
    end
```

Modification 10:
This modification concerned replacing the direction dependent algorithm for calculating the advection and diffusion multipliers with one using the centroid parameters. The following mainline code is the original algorithm:

```
***** Horizontal advection and diffusion multipliers

CDIR$ VECTOR
    DO 10190 F=1,NHQF

******* Positive flows

      IF (LEFTM1_BOUNDARY(F)) BL(ILB(F),QD(F)) = BL(IB(F),QD(F))
      IF (LEFT_FLOWB(F)) THEN
       BL(IB(F),QD(F))  = BL(JB(F),QD(F))
       BL(ILB(F),QD(F)) = BL(JB(F),QD(F))
      END IF
      IF (RIGHT_FLOWB(F)) BL(JB(F),QD(F)) = BL(IB(F),QD(F))
      SF1(F)   = MIN(BL(IB(F),QD(F)),BL(JB(F),QD(F)))
       SF2(F,1)  = BL(IB(F),QD(F))**2
      DEN1(F,1) = 0.25*(BL(ILB(F),QD(F))+2.*BL(IB(F),QD(F))
     .          +BL(JB(F),QD(F)))*(BL(ILB(F),QD(F))
     .          +BL(IB(F),QD(F)))
      DEN2(F,1) = -0.25*(BL(IB(F),QD(F))+BL(JB(F),QD(F)))
     .          *(BL(ILB(F),QD(F))+BL(IB(F),QD(F)))
      DEN3(F,1) = 0.25*(BL(IB(F),QD(F))+BL(JB(F),QD(F)))
     .          *(BL(ILB(F),QD(F))+2.0*BL(IB(F),QD(F))
     .          +BL(JB(F),QD(F)))
      T2(F,1)   = BL(JB(F),QD(F))/(BL(IB(F),QD(F))+BL(JB(F),QD(F)))
      T3(F)     = BL(IB(F),QD(F))/(BL(IB(F),QD(F))+BL(JB(F),QD(F)))
      TP1(F,1)  = 0.5*(BL(IB(F),QD(F))-BL(JB(F),QD(F)))*SF1(F)
      TP2(F,1)  = 0.5*(BL(ILB(F),QD(F))+2.0*BL(IB(F),QD(F))
     .          -BL(JB(F),QD(F)))*SF1(F)
      TP3(F,1)  = 0.5*(BL(ILB(F),QD(F))+3.0*BL(IB(F),QD(F)))*SF1(F)

******* Negative flows

      IF (LEFT_FLOWB(F)) BL(IB(F),QD(F)) = BL(JB(F),QD(F))
      IF (RIGHT_FLOWB(F)) THEN
```

```
        BL(JB(F),QD(F)) = BL(IB(F),QD(F))
        BL(JRB(F),QD(F)) = BL(IB(F),QD(F))
       END IF
       IF (RIGHTP1_BOUNDARY(F)) BL(JRB(F),QD(F)) = BL(JB(F),QD(F))
        SF1(F)   = MIN(BL(IB(F),QD(F)),BL(JB(F),QD(F)))
       SF2(F,2)  = BL(JB(F),QD(F))**2
       DEN1(F,2) = 0.25*(BL(IB(F),QD(F))+2.0*BL(JB(F),QD(F))
     .          +BL(JRB(F),QD(F)))*(BL(IB(F),QD(F))+BL(JB(F),QD(F)))
       DEN2(F,2) = -0.25*(BL(JB(F),QD(F))+BL(JRB(F),QD(F)))
     .          *(BL(IB(F),QD(F))+BL(JB(F),QD(F)))
       DEN3(F,2) = 0.25*(BL(IB(F),QD(F))+2.0*BL(JB(F),QD(F))
     .          +BL(JRB(F),QD(F)))*(BL(JB(F),QD(F))
     .          +BL(JRB(F),QD(F)))
       T1(F)    = BL(JB(F),QD(F))/(BL(IB(F),QD(F))+BL(JB(F),QD(F)))
       T2(F,2)  = BL(IB(F),QD(F))/(BL(IB(F),QD(F))+BL(JB(F),QD(F)))
       TP1(F,2) = -0.5*(3.0*BL(JB(F),QD(F))+BL(JRB(F),QD(F)))*SF1(F)
       TP2(F,2) = 0.5*(BL(IB(F),QD(F))-2.0*BL(JB(F),QD(F))
     .          -BL(JRB(F),QD(F)))*SF1(F)
       TP3(F,2) = 0.5*(BL(IB(F),QD(F))-BL(JB(F),QD(F)))*SF1(F)
      10190 CONTINUE
```

The above code was replaced with the following centroid parameter based algorithm:

```
      DO 10195 F=1,NHQF

******* Positive flows

       IF (LEFTM1_BOUNDARY(F)) BIL(F) = BI(F)
       IF (LEFT_FLOWB(F)) THEN
        BI(F) = 2.0*BID(F)
        BIL(F) = BI(F)
       END IF
       IF (RIGHT_FLOWB(F))THEN
```

```
          BI(F)  = 2.0*BID(F)
          BIR(F) = BI(F)
         END IF
        SF1(F)   = BI(F)
        SF2(F,1)  = BI(F)**2
        DEN1(F,1) = (BIL(F)+BI(F))*BIL(F)
        DEN2(F,1) = -BI(F)*BIL(F)
        DEN3(F,1) =  BI(F)*(BIL(F)+BI(F))
        T2(F,1)  = (BI(F)-BID(F))/BI(F)
        T3(F)    = BID(F)/BI(F)
        TP1(F,1)  = (2.0*BID(F)-BI(F))*SF1(F)
        TP2(F,1)  = (2.0*BID(F)-BI(F)+BIL(F))*SF1(F)
        TP3(F,1)  = (BIL(F)+2.0*BID(F))*SF1(F)

*******  Negative flows

        IF (LEFT_FLOWB(F)) BI(F) = 2.0*BID(F)
        IF (RIGHT_FLOWB(F)) THEN
          BI(F) = 2.0*BID(F)
         BIR(F)  = BI(F)
        END IF
        IF (RIGHTP1_BOUNDARY(F)) BIR(F) = BI(F)
        SF1(F)   =  BI(F)
        SF2(F,2)  =  BI(F)**2
        DEN1(F,2) = (BI(F)+BIR(F))*BI(F)
        DEN2(F,2) = -BI(F)*BIR(F)
        DEN3(F,2) = (BI(F)+BIR(F))*BIR(F)
        T1(F)    = (BI(F)-BID(F))/BI(F)
        T2(F,2)  =  BID(F)/BI(F)
        TP1(F,2)  = -(2.0*(BI(F)-BID(F))+BIR(F))*SF1(F)
        TP2(F,2)  = (2.0*BID(F)-BI(F)-BIR(F))*SF1(F)
        TP3(F,2)  = (2.0*BID(F)-BI(F))*SF1(F)

 10195 CONTINUE
```

Appendix D
MAPPER Input and Output Files for Example Grid

This appendix presents an example grid and the associated input and output files for MAPPER.C. The example consists of a 3 by 2 by 2 grid domain of 12.0 m^3, with element dimensions 1.0 by 1.0 by 1.0 m. The node numbers are shown at the corners and facial mid-points, and the element numbers are indicated in the center of each element in Figure D1. Plan views are shown in Figure D1 for the two layers at the top, middle, and bottom of each layer. The node-element connectivity output file from RC4IC10.f is shown for the example grid in Table D1. This file provides the input required for MAPPER.C. The output files from MAPPER.C, MAP and GEO, are shown for the example grid in Tables D2 and D3, respectively. Following is a description of the steps to conduct this example, along with explanations for the input and output files.

MAPPER Input File Description

The hydrodynamic grid is initially generated by the FastTABS program, which is a preprocessor and postprocessor for two-dimensional (2-D) finite element models maintained by the U.S. Army Engineer Waterways Experiment Station Coastal and Hydarulics Laboratory. The user interactively constructs a grid within FastTABS that then outputs a mesh geometry file for later input into RMA. All information contained in this file is subsequently written by RMA to what is referred to as the RMA binary output file. This RMA output file ultimately will contain a complete geometric description of the grid in 3-D along with the time-varying hydrodynamic flow fields. The RC4IC10.f postprocessing program reads the RMA binary output file and creates the node-element connectivity file, which is the input file for MAPPER.C

The node-element connectivity file (i.e., Table D1) is described as follows. Information in the first section of this file describes the 2-D mesh geometry of the grid. Each line begins with a line type code consisting of two or three letters. Code definitions follow below.

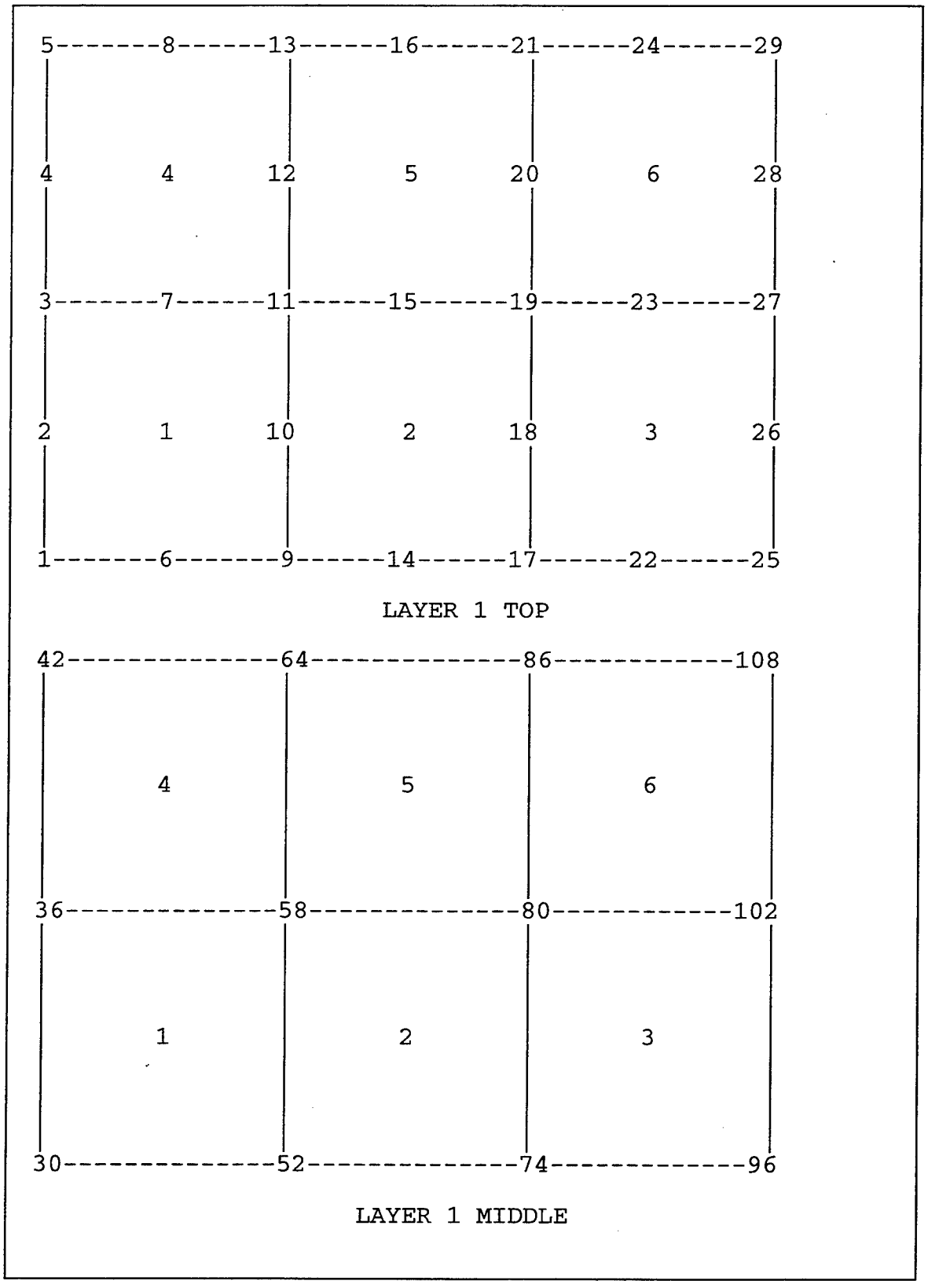


Figure D1. Plan view of example 3 by 1 by 2 grid (Sheet 1 of 3)

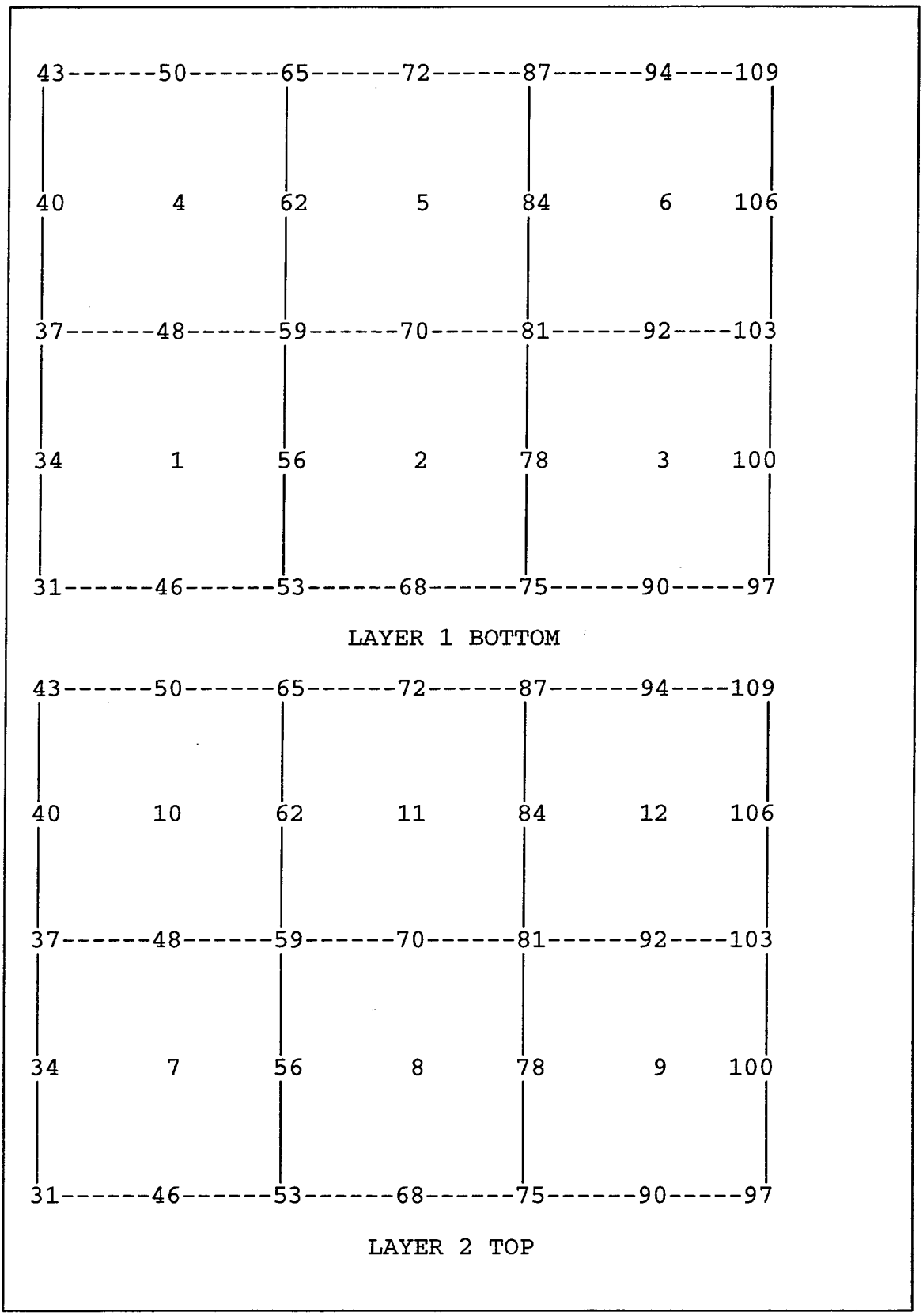


Figure D1. (Sheet 2 of 3)

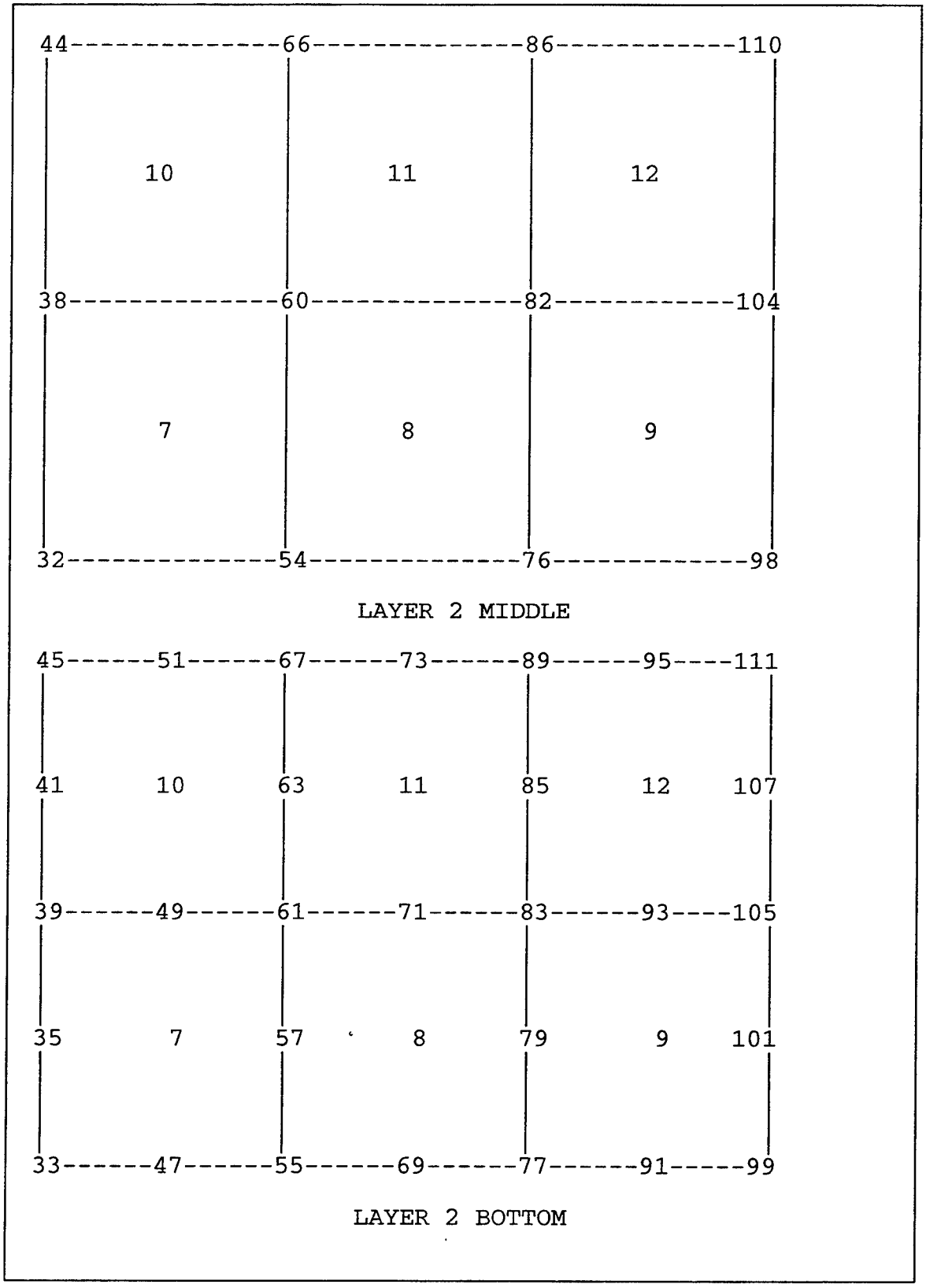


Figure D1. (Sheet 3 of 3)

Table D.1. Node-Element Connectivity File for Example Grid from RC4IC10.f Output File

```
T1 Example node-element connectivity description file
T2
T3
SI  0
$L  3   0   6   0
GO  1   2   3   4   5   -1

GE      1       1       6       9      10      11       7       3       2       1     0.0
GE      4       3       7      11      12      13       8       5       4       1     0.0
GE      2       9      14      17      18      19      15      11      10       1     0.0
GE      5      11      15      19      20      21      16      13      12       1     0.0
GE      3      17      22      25      26      27      23      19      18       1     0.0
GE      6      19      23      27      28      29      24      21      20       1     0.0
GNN     1         1.0         1.0          100.0
GNN     3         1.0         2.0          100.0
GNN     5         1.0         3.0          100.0
GNN     9         2.0         1.0          100.0
GNN    11         2.0         2.0          100.0
GNN    13         2.0         3.0          100.0
GNN    17         3.0         1.0          100.0
GNN    19         3.0         2.0          100.0
GNN    21         3.0         3.0          100.0
GNN    25         4.0         3.0          100.0
GNN    27         4.0         3.0          100.0
GNN    29         4.0         3.0          100.0
VALUES FROM 3D BINARY GEOMETRY FILE (generated by ricr10.f)
    NP     NE    NPM     NEM
   111     12     29      12


  SURFACE
  NODE       NDEP      NREF

     1          5        29
```

```
     2          3        34
     3          5        35
     4          3        39
     5          5        41
     6          3        45
     7          5        47
     8          3        49
     9          5        52
    10          3        55
    11          5        57
    12          3        61
    13          5        63
    14          3        67
    15          5        69
    16          3        71
    17          5        73
    18          3        77
    19          5        79
    20          3        83
    21          5        85
    22          3        89
    23          5        91
    24          3        93
    25          5        96
    26          3        99
    27          5       102
    28          3       105
    29          5       108

NODE   NSURF
   1    1200                    ! 1200 means head boundary strip
  30    1200
  31    1200
  32    1200
  33    1200
```

```
 2    1200
34    1200
35    1200

 3    1200
36    1200
37    1200
38    1200
39    1200

 4    1200
40    1200
41    1200

 5    1200
42    1200
43    1200
44    1200
45    1200

 6    1000                ! 1000 means wall Boundary strip
46    1000
47    1000

 7       0                ! 0 means internal strip
48       0
49       0

 8    1000
50    1000
51    1000

 9    1000
52    1000
53    1000
54    1000
```

```
55  1000
10  0
56  0
57  0
11  0
58  0
59  0
60  0
61  0
12  0
62  0
63  0
13  1000
64  1000
65  1000
66  1000
67  1000
14  1000
68  1000
69  1000
15  0
70  0
71  0
16  1000
72  1000
73  1000
17  1000
74  1000
```

```
75      1000
76      1000
77      1000

18         0
78         0
79         0

19         0
80         0
81         0
82         0
83         0

20         0
84         0
85         0

21      1000
86      1000
87      1000
88      1000
89      1000

22      1000
90      1000
91      1000

23         0
92         0
93         0

24      1000
94      1000
95      1000
```

```
1200
1200
1200
1200
1200

25
96
97
98
99

1200
1200
1200

26
100
101

1200
1200
1200
1200
1200

27
102
103
104
105

1200
1200
1200

28
106
107

1200
1200
1200
1200
1200

29
108
109
110
111
```

Table D.2. MAP File Generated for Example Grid

ICM	FEM
1	1
2	2
3	3
4	4
5	5
6	6
7	7
8	8
9	9
10	10
11	11
12	12

FFID	ILB	IB	JB	JBR	TYPE	ORIENT	TAG	LAYER
1	2	1	0	0	B	H	1	1
2	0	1	2	3	I	H	2	1
3	0	1	4	0	I	H	3	1
4	1	2	3	0	I	H	4	1
5	0	2	5	0	I	H	5	1
6	2	3	0	0	B	H	6	1
7	0	3	6	0	I	H	7	1
8	5	4	0	0	B	H	8	1
9	0	4	5	6	I	H	9	1
10	4	5	6	0	I	H	10	1
11	5	6	0	0	B	H	11	1
12	8	7	0	0	B	H	12	2
13	0	7	8	9	I	H	13	2
14	0	7	10	0	I	H	14	2
15	7	8	9	0	I	H	15	2
16	0	8	11	0	I	H	16	2
17	8	9	0	0	B	H	17	2
18	0	9	12	0	I	H	18	2
19	11	10	0	0	B	H	19	2

```
20        0        10        11        12        I        H        20        2
21       10        11        12         0        I        H        21        2

22       11        12         0         0        B        H        22        2

23        0         7         1         0        I        V        23        2-1
24        0         8         2         0        I        V        24        2-1
25        0         9         3         0        I        V        25        2-1
26        0        10         4         0        I        V        26        2-1
27        0        11         5         0        I        V        26        2-1
28        0        12         6         0        I        V        26        2-1

SB      NVF
 1        1
 2        1
 3        1
 4        1
 5        1
 6        1

SB     VFN LIST...
 1       23
 2       24
 3       25
 4       26
 5       27
 6       28
```

Table D.3. GEO File Generated for Example Grid

SBN	BBN
1	7
2	8
3	9
4	10
5	11
6	12

BU
0
0
0
0
0
0
1
2
3
4
5
6

BIL	BI	BID	BIR	TYPE	LAYER
1.0	0.0	0.5	0.0	B	1
0.0	1.0	0.5	1.0	I	1
0.0	1.0	0.5	0.0	I	1
1.0	1.0	0.5	0.0	I	1
0.0	1.0	0.5	0.0	I	1
1.0	0.0	0.5	0.0	B	1
0.0	1.0	0.5	0.0	I	1
1.0	0.0	0.5	0.0	B	1
0.0	1.0	0.5	1.0	I	1
1.0	1.0	0.5	0.0	I	1
1.0	0.0	0.5	0.0	B	1

```
1
1 1
1 1 1
1 1 1 1
1 1 1 1 1
1 1 1 1 1 1
1 1 1 1 1 1 1
1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1

B
H H
H H H
H H H H
H B H H H
H B H B H H
H H H H H H H
H H H H H H H H
H H H H H H H H H
H B H B H H B H B

0.0
1.0 0.0
0.0 0.0 0.0
0.0 0.0 0.0 0.0
0.0 0.0 0.0 0.0 0.0
0.0 1.0 0.0 0.0 0.0 0.0
0.0 0.0 0.0 0.0 0.0 0.0 0.0
0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
0.0 1.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0

0.5
0.5 0.5
0.5 0.5 0.5
0.5 0.5 0.5 0.5
0.5 0.5 0.5 0.5 0.5
0.5 0.5 0.5 0.5 0.5 0.5
0.5 0.5 0.5 0.5 0.5 0.5 0.5
0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5
0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5
0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5

0.0
1.0 0.0
1.0 1.0 0.0
1.0 1.0 0.0 0.0
1.0 1.0 0.0 0.0 0.0
1.0 1.0 0.0 0.0 0.0 0.0
1.0 1.0 0.0 0.0 0.0 0.0 0.0
1.0 1.0 0.0 0.0 0.0 0.0 0.0 0.0
1.0 1.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
1.0 1.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0

1.0
0.0 1.0
0.0 1.0 1.0
1.0 0.0 1.0 1.0
1.0 0.0 1.0 1.0 1.0
1.0 0.0 1.0 1.0 1.0 1.0
1.0 0.0 1.0 1.0 1.0 1.0 1.0
1.0 0.0 1.0 1.0 1.0 1.0 1.0 1.0
1.0 0.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0
1.0 0.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0
```

T1 thru T3: Titles. T1 and T2 are optional, T3 is necessary.

SI: Signifies English or SI units. English = 0, SI = 1.

$L: logical unit numbers used by RMA specific routines.

GO: Used to input the nodes of a GO string (RMA term). Used as a reference line for a renumbering of node identifier numbers done by RMA internally.

GE: Element definitions.

Field	Description
1	The first integer is the element identifier.
2-9	The next eight integers define the nodal connectivity of the element. The node ordering is counter-clockwise beginning at a corner then sequentially around the perimeter. Eight integers are always used. Triangular elements that have only six nodes will have the remaining two integers set to zero.
10	Material identifier (internal to RMA)
11	A real value. Represents the direction of the eddy viscosity tensor for the element and is usually left as zero.

GNN: Corner node coordinates.

Field	Description
1	Integer. Node identifier.
2	Real. X coordinate.
3	Real. Y coordinate.
4	Real. Z coordinate.

Midside nodes are not listed. They can be calculated from corner nodes.

Information in the second section Table D1 describes the depth dimension of the grid. This section begins with the specification of the following parameters:

NP - Total number of nodes in grid.

NE - Total number of elements in grid.

NPM - Total number of nodes in the surface plane.

NEM - Total number of elements in surface plane.

Next, the number of nodes beneath each surface node, including the surface node, and an integer indicating the numbering of this column of nodes are specified.

NDEP - Total number of nodes in a column. A column exists for each node in the surface plane.

NREF - Nodes in the surface plane are numbered sequentially. However, all other nodes are numbered in sequential order from the node beneath the surface node to the bottom within a node column. This integer is the starting node identifier for the first node beneath the surface plane node in a node column plus 1.

The final section of Table D1 indicates for each node within a node column whether it is internal to the grid or positioned on the boundary. The following definitions apply.

NSURF: An integer associated with each node column indicating the type of node column. Possible values are as follows:

0 - Internal node column.

1000 - Wall boundary exists at this node column.

1200 - Head boundary exists at this node column.

31000 - M2 tidal boundary exists at this node column.

MAPPER Output Files Descriptions

The program MAPPER.C reads the node-element connectivity file and generates the output files MAP and GEO files used as input by ICM. When MAPPER reads the node-element connectivity file, it builds an internal RMA grid. MAPPER is aware of how RMA numbers its nodes and elements. After completing the internal RMA grid, MAPPER can then write out the linkage and geometry data ICM needs. MAPPER will create the MAP and GEO files. The MAP file contains the RMA element to ICM cell correspondence and flow face definitions. The GEO file specifies the vertical relationship between ICM cells and lists the QUICKEST weighting distances.

ICM MAP File Description

The MAP file contains four sections of data. A brief explanation of each section follows below.

The first section lists the correspondence between ICM cell identifiers and RMA element identifiers. Note that the two models have different numbering schemes for their cells. This section establishes the relationship between the two numbering schemes. The FEMCONVT.F program, which reads the RMA binary output file, writes out data in terms of RMA internal element numbers and node numbers; therefore, the information in this section will allow a unique identification of each cell and flow face.

RMA numbers elements sequentially in the order of their creation for the surface layer. Next, it numbers its elements from the bottom up to the layer beneath the surface layer going from one column to the next in the order of the element numbering in the surface layer. The ICM numbering scheme sequentially numbers the cells within a layer starting with the surface layer then progressing to the layer beneath till the last layer has been processed. Definitions of the parameters for this section follow.

ICM - Unique ICM cell identifier.

FEM - Unique RMA cell identifier corresponding to the appropriate ICM cell.

The second section identifies each flow face in the grid. Definitions of the parameters for flow face definition follow.

FFID - Unique flow face identifier.

ILB - QUICKEST far upstream weighting cell identifier.

IB - Cell identifier for flow donor cell.

JB - Cell identifier for flow receiver cell.

JRB - QUICKEST far downstream weighting cell identifier.

TYPE - Character variable indicating type of flow face. B indicates Boundary. I indicates Internal flow face.

ORIENT - Character variable indicating direction orientation. H indicates face is oriented for horizontal flow. V indicates face is oriented for vertical flow.

LAYER - Indicates what layer the flow face is in.

The third section lists the number of vertical flow faces in a cell column beneath each surface cell. Definitions of the parameters for this section follow.

SB - Surface cell (box) identifier.

NVF - Number of vertical flow faces beneath the surface cell.

The fourth section lists the vertical flow face identifiers for each ICM cell column iterated in the third section. Definitions of the parameters necessary for this section follow.

SB - Surface cell (box) identifier.

VFN - List of all flow face identifiers oriented vertically beneath a surface cell.

ICM GEO File Description

The GEO file contains tree sections of data. A brief explanation of each section follows below.

The first section identifies for each surface cell the bottom cell directly beneath it. Definitions of the parameters for this section follow.

SB - Surface cell (box) identifier.

BBN - Bottom cell identifier.

The second section identifies for each cell, in increasing order, the cell directly above it. Definition of the parameter for this section follows.

BU - Identifier of cell above. Cell identifier is implicit in line order.

The third section specifies the QUICKEST weighting distances for each flow face. Definitions of the parameters for this section follow.

BI - Centroid to centroid distance between IB cell JB cell.

BID - Distance from centroid of IB cell to flow face.

BIL - Centroid to centroid distance between ILB cell and IB cell.

BIR - Centroid to centroid distance between JB cell and JRB cell.

TYPE - Character variable indicating type of flow face. B indicates Boundary. I indicates Internal flow face.

LAYER - Indicates what layer the flow face is in.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	August 1997	Final report

4. TITLE AND SUBTITLE
Development of Unstructured Grid Linkage Methodology and Software for CE-QUAL-ICM

5. FUNDING NUMBERS

6. AUTHOR(S)
Raymond S. Chapman, Terry K. Gerald, Mark S. Dortch

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Ray Chapman and Associates, 1725 MacArthur Place, Vicksburg, MS 39180;
AScI Corporation, 3402 Wisconsin Avenue, Vicksburg, MS 39180;
U.S. Army Engineer Waterways Experiment Station, 3909 Halls Ferry Road, Vicksburg, MS 39180-6199

8. PERFORMING ORGANIZATION REPORT NUMBER
Miscellaneous Paper W-97-1

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
U.S. Army Corps of Engineers
Washington, DC 20314-1000

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES
Available from National Technical Information Service, 5285 Port Royal Road, Springfield, VA 22161.

12a. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution is unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT ***(Maximum 200 words)***

This study was conducted for the purpose of developing a methodology and associated software for linking hydrodynamic output from the RMA10 finite element model to the CE-QUAL-ICM finite volume water quality model. The ICM model is usually linked to hydrodynamic output from a finite difference hydrodynamic model, CH3D-WES. RMA10 uses an unstructured grid approach, whereas CH3D-WES is based on a structured gird scheme. A linkage to RMA10 was needed to provide greater flexibility for simulating water quality and contaminants. This report documents the development and testing of the RMA10-ICM linkage.

14. SUBJECT TERMS
Hydrodynamic model
Linkage
Numerical model
Unstructured grids
Water quality model

15. NUMBER OF PAGES
185

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED		